Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

an der Universität Hamburg eingereichte

Cumulative Dissertation

Resilience of Service-oriented and Time-sensitive Mission-critical Networks

Doğanalp Ergenç

Faculty of Mathematics, Informatics, and Natural Sciences
Department of Computer Science
Computer Networks (NET) Group

Hamburg, July 11, 2023

Reviewers

Prof. Dr.-Ing. Mathias Fischer
Dr hab. inż. Jacek Rak
Prof. Dr.-Ing. habil. Falko Dressler

Disputation date: 04.07.2023

Abstract

Modern mission-critical systems (MCSs) have become complex technological ecosystems that consist of several inter-connected services with various quality of service (QoS) and resilience requirements. Their increasing heterogeneity and connectivity make MCSs vulnerable to further safety and security threats. Moreover, traditional system design and networking technologies remain inadequate due to their limitations on configurability and extensibility. Therefore, two new design paradigms, service-oriented architecture (SOA) and IEEE 802.1 Time-sensitive Networking (TSN) protocols, have been recently employed in MCSs to fulfill their evolving requirements. SOA enables allocating service instances on top of virtualized embedded nodes, which provides significant design flexibility. TSN unifies various networking equipment and protocols to forward time-sensitive data streams on top of standard Ethernet technologies that are less costly and easy to deploy. Both paradigms enable novel countermeasures against safety and security threats, thus increasing the resilience of MCSs. However, this induces additional complexity for distributing services and configuring time-sensitive streams, which can introduce additional safety and security threats that should be rigorously investigated.

Accordingly, in this *cumulative* thesis, we present several contributions to collectively build resilient, service-oriented, and time-sensitive MCSs. These contributions constitute the complementary design artifacts for the initial configuration, maintenance, and protection of MCSs. They are further aligned with our primary resilience goals: fault tolerance, autonomy, and security.

For resilient service-oriented design, we first optimally allocate resources for mixed-criticality services and data routing satisfying their QoS requirements. This also includes reserving backup resources for *fault tolerance* in case of node and link failures. Then, we develop a distributed and *autonomous* orchestration mechanism to maintain this service configuration without relying on a centralized controller. Lastly, we model defensive strategies by redistributing services and rerouting their communication paths for an increased *security* against targeted attacks.

For resilient time-sensitive networking, firstly, we optimally schedule time-sensitive streams by developing an *autonomous* configuration mechanism for a prominent TSN scheduling protocol. We then propose a strategy to find the most reliable redundant paths against multiple link failures and configure them for the only *fault tolerance* protocol among TSN standards. Lastly, we explore potential attack vectors against TSN protocols and propose an open-source *security* monitoring system.

Finally, we discuss several potential research topics to address further complexity, interconnectivity, and security issues regarding MCSs as future work.

Zusammenfassung

Moderne missionskritische Systeme (MS) haben sich zu komplexen technologischen Ökosystemen entwickelt, die aus mehreren miteinander verbundenen Diensten mit unterschiedlichen Anforderungen an die Dienstgüte und Resilienz bestehen. Ihre zunehmende Heterogenität und Konnektivität machen sie anfällig für weitere Sicherheitsbedrohungen. Darüber hinaus sind herkömmliche Systemdesign- und Netzwerktechnologien aufgrund ihrer begrenzten Konfigurierbarkeit und Erweiterbarkeit nach wie vor unzureichend. Daher werden in MS seit kurzem zwei neue Entwurfsparadigmen eingesetzt, um die sich entwickelnden Anforderungen zu erfüllen: Serviceorientiere Architektur (SOA) und IEEE 802.1 Time-sensitive Networking (TSN)-Protokolle. SOA ermöglicht die Zuweisung von Serviceinstanzen auf virtualisierte eingebettete Netzwerkknoten, was eine erhebliche Designflexibilität bietet und die Rekonfiguration im Falle eines Ausfalls oder Angriffs ermöglicht. TSN vereinigt verschiedene Netzwerkgeräte und -protokolle, um zeitkritische Datenströme auf der Grundlage von Standard-Ethernet-Technologien weiterzuleiten, die weniger kostspielig und einfach zu implementieren sind. Beide Paradigmen ermöglichen neuartige Gegenmaßnahmen gegen Sicherheitsbedrohungen und erhöhen so die Resilienz von MSe. Dies führt jedoch zu zusätzlicher Komplexität bei der Verteilung von Diensten und der Konfiguration zeitkritischer Datenströme, was zusätzliche Sicherheitsbedrohungen mit sich bringen kann, die gründlich untersucht werden sollten.

Dementsprechend stellen wir in dieser Arbeit mehrere Beiträge vor, die es zusammen ermöglichen, resiliente, serviceorientierte sowie zeitkritische MS zu konstruieren. Diese Beiträge bilden die ergänzenden Design-Artefakte für die anfängliche Konfiguration, Wartung und den Schutz von MSe. Weiterhin sind sie nach unseren primären Resilienzzielen ausgerichtet: Fehlertoleranz, Autonomie und Sicherheit.

Für ein belastbares serviceorientierts Design weisen wir zunächst die Ressourcen für Dienste mit gemischter Kritikalität und für das Datenrouting optimal zu, sodass Anforderungen an die Dienstgüte erfüllt werden. Dazu gehört auch die Reservierung von Backup-Ressourcen für *Fehlertoleranz* im Falle von Knoten- und Verbindungsausfällen. Dann entwickeln wir einen verteilten und *autonomen* Orchestrierungsmechanismus, um diese Servicekonfiguration zu erhalten, ohne auf einen zentralen Controller angewiesen zu sein. Abschließend modellieren wir defensive Strategien, indem wir Dienste umverteilen und ihre Kommunikationspfade umleiten, um die Sicherheit gegen gezielte Angriffe zu erhöhen.

Für resiliente zeitkritische Netzwerke planen wir zunächst zeitkritische Streams optimal, indem wir einen *autonomen* Konfigurationsmechanismus für ein bekanntes Scheduling-Protokoll entwickeln. Dann schlagen wir eine Strategie vor, um die zuverlässigsten redundanten Pfade gegen mehrere Verbindungsausfälle zu finden und sie für das einzige *Fehlertoleranz*-Protokoll unter den TSN-Standards zu konfigurieren. Schließlich untersuchen wir potenzielle Angriffsvektoren gegen TSN-Protokolle und schlagen ein open-source *Sicherheit*ssystem basierend auf Monitoring vor.

Abschließend erörtern wir mehrere potenzielle Forschungsthemen, um weitere Komplexitäts- und Sicherheitsprobleme in Bezug auf MSe als zukünftige Arbeit aufzuzeigen.

Acknowledgement

They say PhD is like a marathon than a sprint: You have to run steadily with great discipline over a long time. For me, it takes beyond endurance and dedication as it exposes you to endless, various challenges that one could not naturally suffice or anticipate. It is definitely not trivial to prevail alone.

Therefore, I would like to express my gratitude to the people who supported me during these four years of continuous challenge. Firstly, and maybe most importantly, Prof. Mathias Fischer has been a great mentor and also a very good friend to me throughout the journey. He has not only taught me a lot about science and academia; we have laughed, played, enjoyed, and literally run a marathon together. Prof. Jacek Rak helped and inspired me to develop new and exciting ideas from the very beginning of my PhD. Prof. Ertan Onur pulled me into academia and immensely encouraged me. He has always been there whenever I need his wisdom. They have surely become the most inspiring figures in my academic life.

I feel fortunate to have all the great people of my research group running this marathon beside me. Steffen warmly welcomed me to the team when I was a total stranger in a new university - and a new country. August's cheerful and easy-going vibe made everything much more bearable. With Nurefsan, we have found our research path together. I have always admired her endurance and ambition, and owe her some papers to be written together. Thanks to Pascal, I had never felt alone in the office when the whole campus was still empty after depressing quarantine days. Last but not least, we have poured out our grieves to each other with Florian in both our academic and personal lives. He is one of the funniest, smartest, and most empathetic persons I have ever known.

Finally, I would like to thank those people who inspirited me ceaselessly. I could not do it without the unconditional support of my family. I am really sorry to make my mom concerned about my health and sanity all the time, and I sincerely appreciate my dad's immense effort to calm her down. My best friends, Aslıhan, Burcu, Can, Cem, Mert, Tuğce, and Yağmur, have heartened me for the last ten years. I cannot apologize enough for dozens of times when I could not pick up their phone calls excusing paper deadlines. Ceren, my dearest, has been my sunshine at the darkest times of this journey. She made me feel much stronger and loved with her huge support and understanding in the last and by far the hardest year of my PhD. Lastly, although she probably cannot read this, I should mention my adorable, fluffy companion, Fındık. Adopting her was undoubtedly one of the best decisions of my life.

At the end of the run, one might feel exhausted and weariful, but it is absolutely worth all the sense of accomplishment. I hope this thesis could inspire and enlighten the journey of many other young researchers like myself.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation & Problem Statement

Modern mission-critical systems (MCSs) such as autonomous vehicles, avionics, and industrial networks are complex technological ecosystems. They are composed of various critical and non-critical services that should communicate with strict quality of service (QoS) requirements. For instance, recent autonomous cars from Mercedes-Benz are equipped with more than 30 sensors [Mer22] for high-precision environmental data processed for making potentially critical decisions in real-time. Likewise, with Industry 4.0, intelligent cyber-physical systems evolve with collaborating robotics and embedded nodes [ZXKN17, Koe18] that require reliable and time-sensitive communication. These MCSs might also be connected to others to form, for example, complex vehicular networks or extended industrial facilities with even more intricate and interdependent services. Most importantly, several interconnected services perform critical tasks and thus should be resilient against potential failures or attacks that can easily lead to hazardous safety and security consequences.

While MCSs have been evolving with increasing heterogeneity, connectivity, and resulting complexity, traditional system design and communication paradigms become inadequate to satisfy their QoS and resilience requirements. Legacy system architectures and communication protocols accordingly pose the following major problems:

Problem 1. Static nodes with dedicated functions: Traditional MCSs are statically-designed distributed networks of embedded nodes. In MCSs, each system node has a dedicated function and is implemented with tightly-coupled hardware and software modules. This static design brings a substantial manual effort to reconfigure the system functions as they enforce the rewiring of the physical nodes. Accordingly, it severely limits the configurability and extensibility of MCSs. Such limitations also lead to further problems, such as a lack of adaptability in the case of emerging resource requirements.

Problem 2. Lack of standardized network components: MCSs typically employ tailored networking equipment and legacy protocols. For instance, while those protocols are Controller Area Network (CAN), Local Interconnected Network (LIN), and Media Oriented System Transport (MOST) [TGH+15] in automotive, Avionics Full Duplex Ethernet (AFDX) [SG12] addresses very similar communication needs in avionics. The result of such diversity across domains is a lack of generic networking equipment and protocols. This constraints design, evaluation, and improvement of MCSs to a limited number of technology providers, which significantly increases their design cost. Besides, potential technology transfers between different domains remain challenging.

Problem 3. Limited and static resilience countermeasures: The limitations of the system and network components also restrict the design of resilience countermeasures. For instance, legacy MCSs usually deploy duplicate static components, e.g., cold and hot backups, redundant links, with diverse software and hardware architectures for *fault tolerance* via redundancy. While this approach increases the cost and complexity of their design and maintenance, it could be effective only against predictable faults. Besides, all those active and redundant nodes and their intercommunication are orchestrated by a centralized controller, or even manually configured. It reduces (re-)configurability and *autonomy* of MCSs and hinders quick reactions, e.g., local failovers, against safety and security incidents. A strong dependency on a centralized controller also induces a risk of a single point of failure.

Lastly, since they were often closed-loop systems with restricted connectivity, MCSs are equipped with limited *security* mechanisms s.t. relatively simple monitoring and logging facilities for their domain-specific network protocols and static system nodes. On the contrary, increasing connectivity and complexity render them attractive targets for well-calibrated and persistent attacks. Traditional security approaches of MCSs remain ineffective against such threats.

Accordingly, the complexity arising due to the diverse requirements and increasing connectivity of several services requires embracing novel system architecture and networking approaches. While those approaches should ease the design and configuration of MCSs with formidable QoS requirements, they should also guarantee the resilience of MCSs against any safety and security incidents.

1.2 Proposals & Research Questions

To address the problems presented above, we[1] propose the use of two new design paradigms for developing next-generation resilient MCSs. The first paradigm is the **service-oriented architecture (SOA)**, which tackles the design inflexibility of MCSs and their static configuration (Problem 1). The second paradigm is the employment of new **IEEE 802.1 Time-sensitive Networking (TSN)** protocols to address the lack of unified networking technologies (Problem 2).

[1]For better readability, the author refers to himself as *we* in the remainder of the thesis. His main contributions are explicitly stated in case of collaboration with other authors.

We aim to utilize these paradigms to develop advanced resilience countermeasures for an increased fault tolerance, autonomy, and security of modern MCSs (Problem 3). **Fault tolerance** is one of the prominent requirements of a reliable MCS. It has been a part of even early design processes over decades as such systems cannot tolerate more than a few minutes of disruption in a year, i.e., five to seven nines of availability [Cur20]. **Autonomy** gives a system the capability of self-configuration so that a MCS can at least perform its most critical functions in case of any failures and attacks. It requires the deployment of advanced self-(re)configuration procedures adding up further complexity to the system in exchange for increased system resilience [DHM+19]. Lastly, **security** is one of the most significant yet usually neglected aspects of the design of MCSs. Especially with their increasing connectivity, complexity, and heterogeneity, they have become an attractive target of several sophisticated attacks [MKK+21]. Hence, new threat prevention, detection, and mitigation techniques should be developed using novel technologies.

Accordingly, in the remaining of this section, we present (i) the details regarding the utilization of SOA and TSN for mission-critical networks (tackling Problem 1 and 2), (ii) the main research questions regarding their utilization for the resilience goals above (regarding Problem 3), and (iii) an example scenario to illustrate the benefits of the coexistence of those novel paradigms in terms of resilience.

Service-oriented Architecture

Virtualization technologies are widely adopted in different mission-critical domains to replace their rather static and inflexive system architecture (see Problem 1 above). This trend can be observed in different application scenarios such as modern avionics [ZAL19], smart cities [Con18], and future autonomous driving [BDDK20] that require the flexible deployment of services and functions in embedded systems. Beyond that, there are ongoing standardization activities such as Automotive Virtual Platform Specification [GEN20] and Future Airborne Capability Environment (FACE) [The20] to introduce open-source virtualization technologies to the critical in-vehicle networks.

Virtualization eventually enables service-oriented architecture (SOA) [SL18, KW19, VMN21] that leverages virtualized system nodes to host multiple isolated services [NHAM+18]. It allows flexible deployment of inter-connected services with varying QoS requirements on virtualized physical networks. SOA also offers a significant configurability to dynamically redistribute services and re-establish their intercommunication in case of failures and attacks.

However, SOA also brings considerable complexity to find the optimal service configuration within limited resources. This configuration includes allocating services on virtualized nodes with limited processing powers and finding end-to-end routes with sufficient bandwidth between any nodes hosting inter-connected services.

In addition, those paths should fulfill the strict QoS requirements for time-sensitive communication. Beyond the deployment of virtual services, it is also required to reserve sufficient resources for redundant instances and their data routing in-between against potential node and link failures. It reduces the cost compared to having a dedicated physical backup and enables service migrations in case of multiple failures. Accordingly, the following question arises regarding the design of a **fault-tolerant** SOA:

`RQ.S1`**: How to deploy inter-connected services within the limited node and link resources by ensuring fault tolerance against potential failures?**

After the initial service configuration of active and redundant services, the system should be able to modify this configuration **autonomously**, especially to mitigate safety and security incidents quickly. This includes the methods to minimize its dependency on an administrator or a (logically centralized) controller that may also become a single point of failure. As a result, it brings the following question:

`RQ.S2`**: How to deploy, maintain, and reconfigure mixed-criticality services autonomously with minimal dependency on a controller?**

Lastly, MCSs are typically targets of advanced threats, potentially including several calibrated attacks that take extended periods to develop and conduct but can have a devastating impact. The flexibility and reconfigurability of SOA enable service redistribution and data rerouting as mitigation and recovery techniques against such well-calibrated and persistent attacks. This raises the following question regarding the development of **security** strategies leveraging SOA:

`RQ.S3`**: How to develop defensive strategies to protect critical services against persistent attacks by leveraging the flexibility of service-oriented architecture?**

These questions lead to building resilient service-oriented MCSs by discovering the principles of (i) fault-tolerant configuration of services (`RQ.S1`), (ii) their autonomous orchestration (`RQ.S2`), and (iii) their security by dynamic reconfiguration (`RQ.S3`).

Time-sensitive Networking

To overcome the challenges due to the lack of standardized network components (see Problem 2 above), IEEE 802.1 Time-Sensitive Networking (TSN) standards have recently been proposed by the IEEE 802.1 Working Group. They address real-time and deterministic communication requirements of time-sensitive and safety-critical systems on top of the IEEE 802.3 Ethernet standard [IEE17a]. TSN offers several protocols to manage different traffic classes, ensure deterministic communication within a bounded delay, define filtering and networking policies, and improve communication reliability by using redundant paths. Moreover, TSN allows the use of commodity-off-the-shelf Ethernet switches for both best-effort and time-sensitive critical communication simultaneously. As a result, it accelerates the development of mission-critical networks by allowing the use of well-known Ethernet technology.

Since TSN protocols are relatively new, we still need to explore the principles for their optimal and resilient deployment on MCSs. They first require an attentive configuration with zero tolerance for any unexpected jitter. While it was relatively easier in traditional closed-loop and static networks, modern MCSs should handle hundreds of mixed-criticality data streams **autonomously** and adapt to changing conditions, e.g., due to safety and security incidents, mobile system nodes, etc. This brings the following question regarding the self-configuration of time-sensitive streams:

`RQ.T1`: **How to configure mixed-criticality and time-sensitive streams autonomously, satisfying their strict latency requirements?**

Beyond configuring time-sensitive communication, it is required to ensure the uninterrupted operation of critical streams in case of failures. For **fault tolerance**, TSN offers significant configuration flexibility for redundant communication, which enables seamless recovery after node and link failures. However, this flexibility also induces complexity to guarantee the intended degree of redundancy against various failure scenarios. Accordingly, the following question arises:

`RQ.T2`: **How to configure redundant communication reliably for time-sensitive streams?**

Lastly, employing (new) TSN protocols broaden the attack surface of MCSs by potentially inducing new threats. Besides, their increasing connectivity eases the execution of attacks against TSN protocols. Consequently, this raises the following question related to designing **security** countermeasures against novel threats towards TSN protocols:

`RQ.T3`: **How to protect different mission-critical domains against new threats introduced by TSN protocols?**

These questions lead to establish resilient TSN communication in MCSs by discovering the principles of (i) autonomous and dynamic configuration of time-sensitive streams (`RQ.T1`), (ii) their seamless operation in case of failures (`RQ.T2`), and (iii) their security monitoring and protection (`RQ.T3`).

An example scenario for the coexistence of SOA and TSN

To elaborate further on the employment of the proposed paradigms, an envisioned service-oriented and time-sensitive automotive system is illustrated in Figure 1.1. In this model, various services are hosted at virtualized nodes (grey blocks). Control services (red, wheel icon) orchestrate powertrain and chassis domain functions, e.g., brakes and engine control. Peripheral services (green, camera icon) collect environmental data via sensors for collision avoidance. While telematics services (yellow, speaker icon) are for in-car infotainment, management services (blue, gear icon) offer driver assistance that manages body functions, e.g., controlling door locks automatically. Note that some services, e.g., collision avoidance and driver assistance, are hosted at the same node. The coexistence of critical services (e.g., control and management services) and non-critical services (e.g., telematics services) requires considering their varying QoS and resilience requirements.

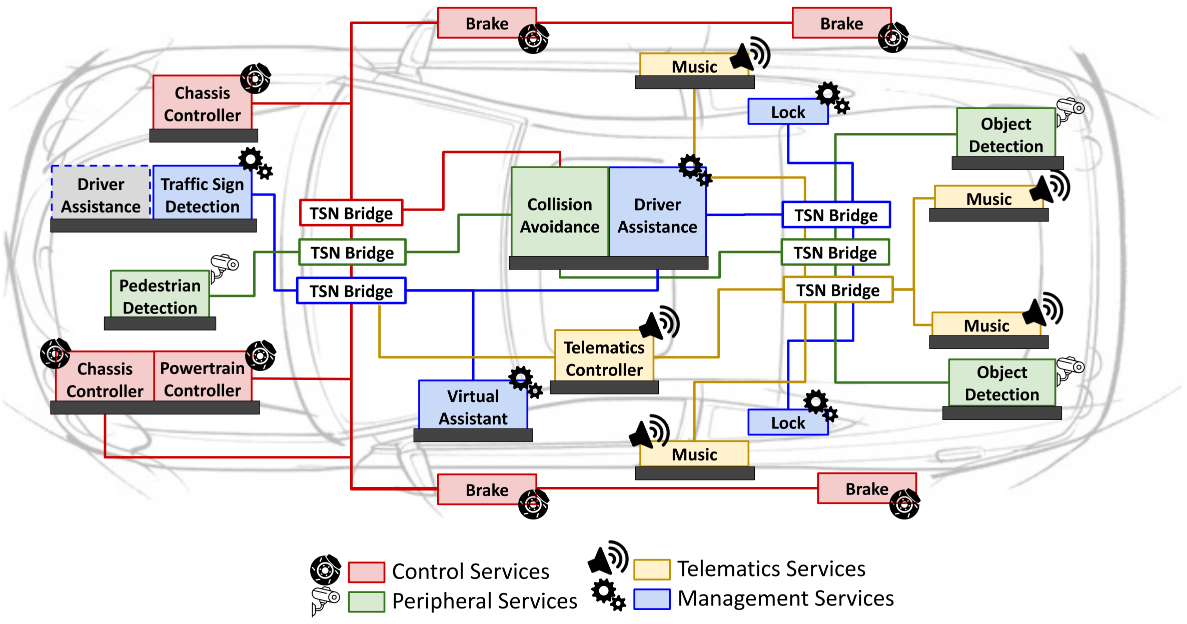


Figure 1.1: An in-vehicle network consisting of different types of services: Control services (red, wheel icon), peripheral services (green, camera icon), telematics services (yellow, speaker icon), and management services (blue, gear icon).

Furthermore, most of the services shown in Figure 1.1 should communicate to perform certain system functions. To realize a critical automatic braking function, for instance, the object detection service should recognize obstacles in real-time and then signal the collision avoidance service hosted at a different node. Depending on the avoidance decision, it might further initiate an emergency brake by communicating an instance of chassis controller service. Moreover, for infotainment, multiple music services can receive data streams from a single telematics controller.

In those examples, while the intercommunication for the automatic braking function requires highly time-sensitive and reliable transfer of event-driven and small amounts of signaling data, the infotainment system requires more bandwidth for continuous streaming with relaxed QoS. TSN protocols enable the coexistence of mixed-criticality streams on unified networking equipment and establish proper resource provision for end-to-end deterministic communication.

Moreover, SOA and TSN together enable several novel resilience mechanisms. The service-based design allows dynamic reconfiguration by reinitiating redundant services or migrating services between nodes in case of a node failure [OSK+19, KJS19]. For example, in Figure 1.1, the redundant driver assistance service instance (grey, on the top left) can be activated if the primary service instance (in the middle) fails for any reason. However, it requires implementing effective service allocation, rerouting, and maintenance routines, which also introduce further complexity to the system. Similarly, TSN standards provide an important versatility for reliable communication by enabling an arbitrary degree of redundancy for the selected mixed-criticality streams. It further ensures deterministic packet delivery in MCSs in which a loss or latent could be easily disruptive. However, they also require a significant configuration effort for their optimal deployment, and the overall complexity broadens the attack and failure surface.

1.3 Contributions

Accordingly, this thesis answers the stated research questions to discover the principles to build resilient service-oriented and time-sensitive MCSs. It specifically considers three resilience goals: fault tolerance, autonomy, and security for the design, configuration, and orchestration of MCSs. This section presents our contributions to (i) resilient service allocation and routing, and (ii) resilient time-sensitive networking regarding SOA and TSN, respectively. Figure 1.2 summarizes them concerning the given resilience goals and the design paradigms. *The complete list of our publications related to the following contributions is also given at the end of this chapter.*

Resilient Service Allocation and Routing

Our contributions to resilient service allocation and routing are (i) fault-tolerant initial service distribution, (ii) an autonomous controlling scheme for service or-

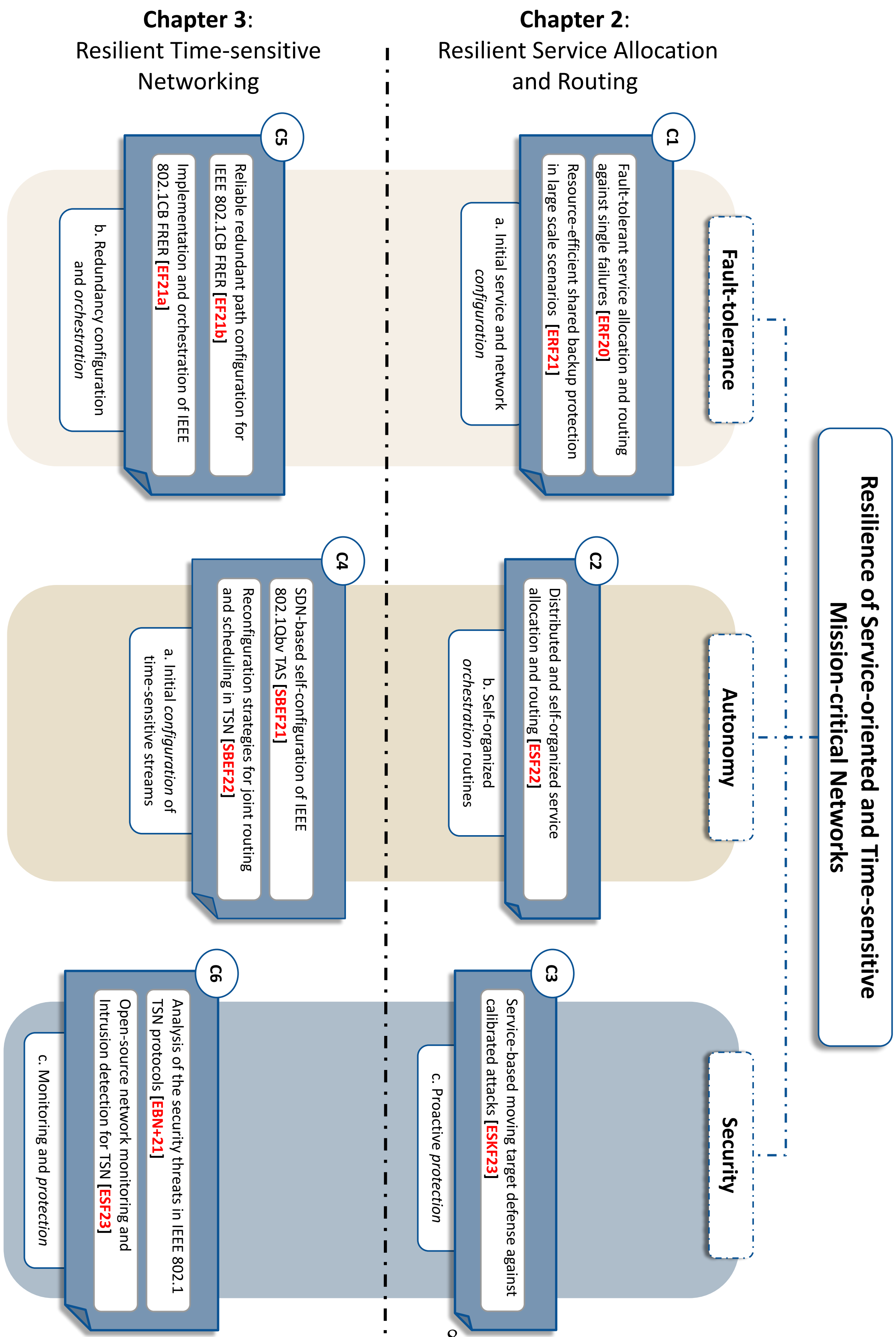

Resilience of Service-oriented and Time-sensitive Mission-critical Networks
Fault-tolerance
Autonomy
Security
Chapter 2: Resilient Service Allocation and Routing
Chapter 3: Resilient Time-sensitive Networking
C1
Fault-tolerant service allocation and routing against single failures [ERF20]
Resource-efficient shared backup protection in large scale scenarios [ERF21]
a. Initial service and network configuration
C5
Reliable redundant path configuration for IEEE 802.1CB FRER [EF21b]
Implementation and orchestration of IEEE 802.1CB FRER [EF21a]
b. Redundancy configuration and orchestration
C2
Distributed and self-organized service allocation and routing [ESF22]
b. Self-organized orchestration routines
C4
SDN-based self-configuration of IEEE 802.1Qbv TAS [SBEF21]
Reconfiguration strategies for joint routing and scheduling in TSN [SBEF22]
a. Initial configuration of time-sensitive streams
C3
Service-based moving target defense against calibrated attacks [ESKF23]
c. Proactive protection
C6
Analysis of the security threats in IEEE 802.1 TSN protocols [EBN+21]
Open-source network monitoring and Intrusion detection for TSN [ESF23]
c. Monitoring and protection

Figure 1.2: The main contributions of the thesis.

chestration, and (iii) dynamic reconfiguration strategies for service protection.

C1. **Fault-tolerant Joint Service Allocation and Routing with Shared Protection:** This contribution handles resource reservation for services and data traffic for the initial fault-tolerant configuration of the system. Firstly, to address the RQ.S1, we propose a set of mixed-integer linear optimization programming (MILP) models that optimally allocate mixed-criticality services as well as redundant instances against random failures in [ERF20] **(Appendix A)**. They simultaneously establish intercommunication between services over multiple redundant paths under tight QoS requirements and limited system resources. Since these models reveal the complexity of the joint service allocation and routing problem, we further extend them with capacity-sharing models in [ERF21] **(Appendix B)** that saves backup resources and allows allocating more services with better QoS.

C2. **Bio-inspired and Autonomous Service Orchestration:** After establishing the initial service allocation and data routing, this contribution provides an autonomous orchestration routine that can be used for reconfiguration of the service deployment. We propose a distributed and bio-inspired joint service allocation and routing model to answer the RQ.S2 in [ESF22] **(Appendix C)**. This model utilizes the ant colony optimization [DG97] to enable self-configuration and minimize the dependency on a controller. Developing this model, we also show the pros and cons of a distributed and autonomous orchestration scheme compared to the centralized approaches.

C3. **Moving Target Defense for Service-oriented Mission-critical Systems:** Lastly, this contribution constitutes a protection layer on top of the initial configuration of critical services and streams by leveraging their reconfigurability in SOA. We combine our previous service distribution models with a theoretical attacker-defender game to develop moving target defense (MTD) strategies in [ESKF23] **(Appendix D)** and to answer the RQ.S3. These strategies enable reconfiguring service allocation and rerouting service communication within a schedule to protect MCSs against calibrated and persistent attacks.

Resilient Time-sensitive Networking

Our contributions to resilient time-sensitive networking include (i) autonomous stream scheduling for initial time-sensitive communication, (ii) reliable redundant path finding and configuration for fault tolerance, and (iii) security monitoring and intrusion detection for time-sensitive data traffic.

C4. **Dynamic Self-(re-)configuration of IEEE 802.1Qbv TAS:** This contribution handles the initial configuration of time-sensitive communication and also tackle dynamically changing traffic requirements. In order to answer the RQ.T1, we propose an optimization model to configure the primary time-sensitive scheduling protocol, IEEE 802.1Qbv Enhancements to Traffic Scheduling: Time-Aware Shaper (TAS), for resource reservation and packet scheduling. This model is then integrated into a Software-defined Networking (SDN)-based self-configuration scheme in [SBEF21] **(Appendix E)** to automatically configure time-sensitive streams without participation of TSN endpoints. We lastly develop various self-configuration strategies in [SBEF22] **(Appendix F)** by using our optimization model.

`C5`**. Reliable Path Finding and Orchestration for IEEE 802.1CB FRER:** On top of initial scheduling of time-sensitive streams, this contribution handles configuration and orchestration of redundant communication. First, we propose a path finding strategy based on a novel graph metric to allocate the most reliable redundant paths for the fault tolerance protocol, IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER), to answer the `RQ.T2`. We also improve its internal functions to avoid potential degradation in redundant communication in [EF21b] **(Appendix G)**. Then, we propose a configuration framework including control plane functions for path discovery and assignment for FRER in [EF21a] **(Appendix H)**.

`C6`**. Security Monitoring and Intrusion Detection for IEEE 802.1 TSN Protocols:** The last contribution provides a security analysis and protection of several TSN protocols. We first explore more than 30 potential attack vectors against all TSN standards and categorize them under a security framework in [EBN+21] **(Appendix I)**. Then, we develop an open-source monitoring and intrusion detection system (IDS) against the explored attack vectors in [ESF23] **(Appendix J)** to answer the `RQ.T3`.

Further Contributions

Apart from the main contributions listed above, we proposed further solutions for the resilient design of next-generation MCSs. For instance, we examine state-of-the-art deep learning techniques for detecting advanced attacks on industrial control systems in [GEF22]. We then discover the several uses of programmable network components to develop reliable monitoring and intrusion prevention mechanisms at in-line processing speed in the [SEO21], [SEO22], and [MSBEF21]. We also present the general takeaways, potential use cases, and future research directions on the security, fault detection, and recovery of virtualized service-oriented MCSs, and the reconfigurability of time-sensitive networks in [EF20] and [BEF22]. Lastly, in [EBF23], we present a prototype combining SOA and TSN protocols to demonstrate different techniques to tackle failures and attacks leveraging these paradigms. We exclude the details of these contributions to present a concise storyline through the main contributions of this thesis.

1.4 Thesis Organization

In the remaining of this *cumulative* thesis, we present our publications in the context of resilient service-oriented and time-sensitive mission-critical networks. We describe the core ideas, the key results, and the takeaways of each publication under the respective sections that are divided according to our resilience goals, i.e., fault tolerance, autonomy, and security. We rephrase and shorten many parts of the publications for a more coherent text, and *all original publications are given in the appendices*. Finally, the overall organization of the thesis is shown in Table 1.1.

Chapter 2 presents our contributions to resilient service allocation and routing. It first introduces the fundamentals of joint service allocation and routing problem and a formal problem definition in Section 2.1. Section 2.2 presents the details of the contribution `C1` regarding fault-tolerant SOA. Section 2.3 describes the dynamics of the self-organized service configuration in the context of `C2`. Lastly, our moving target defense model in `C3` is presented in Section 2.4.

Table 1.1: The organization of the thesis.

Chapter	Section		Contribution	Research Question	Publication
Chapter 1: Introduction					
Chapter 2: Resilient Service Allocation and Routing	Section 2.2	Section 2.2.1	`C1`	`RQ.S1`	[ERF20]
		Section 2.2.2			[ERF21]
	Section 2.3		`C2`	`RQ.S2`	[ESF22]
	Section 2.4		`C3`	`RQ.S3`	[ESKF23]
Chapter 3: Resilient Time-sensitive Networking	Section 3.2	Section 3.2.1	`C4`	`RQ.T1`	[SBEF21]
		Section 3.2.2			[SBEF22]
	Section 3.3	Section 3.2.2	`C5`	`RQ.T2`	[EF21b]
		Section 3.3.2			[EF21a]
	Section 3.4	Section 3.4.1	`C6`	`RQ.T3`	[EBN$^+$21]
		Section 3.4.2			[ESF23]
Chapter 4: Conclusion					

Chapter 3 gives our contributions to resilient time-sensitive networking in detail. Section 3.1 briefly introduces the related TSN protocols that are used and improved in the following contributions. Section 3.2 presents the details of the optimal TAS scheduling and TSN self-configuration scheme in the context of `C4`. In Section 3.3, we present reliable redundant path finding and configuration methods for FRER to establish fault-tolerant communication, regarding `C5`. Lastly, Section 3.4 introduces several novel security threats against TSN protocols and the implementation details of our open-source monitoring and intrusion detection module regarding `C6`.

Chapter 4 summarizes all contributions and presents further research directions.

Main Contributions

[EBN+21] **Doğanalp Ergenç**, C. Bruelhart, J. Neumann, L. Krueger, and M. Fischer. On the Security of IEEE 802.1 Time-Sensitive Networking. *IEEE International Conference on Communications (ICC), Workshop on Time-sensitive and Deterministic Networking*, 2021.

[EF21a] **Doğanalp Ergenç** and M. Fischer. Implementation and Orchestration of IEEE 802.1CB FRER in OMNeT++. *IEEE International Conference on Communications (ICC), Workshop on Time-sensitive and Deterministic Networking*, 2021.

[EF21b] **Doğanalp Ergenç** and M. Fischer. On the Reliability of IEEE 802.1CB FRER. *IEEE International Conference on Computer Communications (INFOCOM)*, 2021.

[ERF20] **Doğanalp Ergenç**, J. Rak, and M. Fischer. Service-Based Resilience for Embedded IoT Networks. *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020.

[ERF21] **Doğanalp Ergenç**, J. Rak, and M. Fischer. Service-based Resilience via Shared Protection in Mission-critical Embedded Networks. *IEEE Transactions on Network and Service Management (TNSM), Special Issue on Design and Management of Reliable Communication Networks*, 2021.

[ESF22] **Doğanalp Ergenç**, D. Sorejevic, and M. Fischer. Distributed Bio-inspired Configuration of Virutalized Mission-critical Networks. *IEEE Global Communications Conference (GLOBECOM)*, 2022.

[ESF23] **Doğanalp Ergenç**, R. Schenderlein, and M. Fischer. TSNZeek: An Open-source Intrusion Detection System for Time-sensitive Networking. *IFIP Networking Conference, International Workshop on Time-Sensitive and Deterministic Networking (TENSOR)*, 2023.

[ESKF23] **Doğanalp Ergenç**, F. Schneider, P. Kling, and M. Fischer. Moving Target Defense for Service-oriented Mission-critical Networks. *International Conference on Computer Communications and Networks (ICCCN)*, 2023.

[SBEF21] N. Sertbaş Bülbül, **Doğanalp Ergenç**, and M. Fischer. SDN-based Self-Configuration for Time-Sensitive IoT Networks. *International Conference on Local Computer Networks (LCN)*, 2021.

[SBEF22] N. Sertbaş Bülbül, **Doğanalp Ergenç**, and M. Fischer. Towards SDN-based Dynamic Path Reconfiguration for Time-sensitive Networking. *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022.

Further Contributions

[BEF22] N. Sertbaş Bülbül, **Doğanalp Ergenç**, and M. Fischer. Evaluating Dynamic Path Reconfiguration for Time-sensitive Networks. *Würzburg Workshop on Next-Generation Communication Networks (WueWoWas'22)*, 2022.

[EBF23] **Doğanalp Ergenç**, C. Brülhart, and M. Fischer. Towards Developing Resilient and Service-oriented Mission-critical Systems. *9th IEEE International Conference on Network Softwarization (NetSoft), Demo Session*, 2023.

[EF20] **Doğanalp Ergenç** and M. Fischer. Resilience of Virtualized Embedded IoT Networks. *2. KuVS Fachgespräch "Network Softwarization"*, 2020.

[GEF22] P. Gawehn, **Doğanalp Ergenç**, and M. Fischer. Deep Learning-based Multi-PLC Anomaly Detection in Industrial Control Systems. *IEEE Global Communications Conference (GLOBECOM)*, 2022.

[MSBEF21] M. Mönnich, N. Sertbaş Bülbül, **Doğanalp Ergenç**, and M. Fischer. Mitigation of IPv6 Router Spoofing Attacks with P4. *ACM Symposium on Architectures for Networking and Communications Systems (ANCS), EuroP4 Workshop*, 2021.

[SEO21] G. Simşek, **Doğanalp Ergenç**, and E. Onur. Efficient Network Monitoring via In-band Telemetry. *17th International Conference on the Design of Reliable Communication Networks (DRCN)*, 2021.

[SEO22] G. Simşek, **Doğanalp Ergenç**, and E. Onur. Reliable and Distributed Network Monitoring via In-band Network Telemetry. *arXiv*, 2022. `https://arxiv.org/abs/2212.14876` (preprint).

Chapter 2

Resilient Service Allocation and Routing

This chapter presents the main contributions of the thesis on resilient service allocation and routing to design service-oriented mission-critical systems (MCSs). Section 2.1 briefly introduces preliminaries on the service-oriented architecture (SOA) and the Joint Service Allocation and Routing (`JSAR`) as a MILP model. We then utilize and extend this model over the following contributions in this chapter. Afterwards, the rest of the sections are organized to present sequential building blocks for developing a resilient service-oriented architecture as follows.

- In Section 2.2, the `JSAR` model is extended with **fault tolerance** constraints to deploy redundant service instances and assign backup paths to tackle single node and link failures [ERF20]. It is further combined with another model to share backup link capacity between several services for resource efficiency [ERF21]. Those contributions address the initial service configuration of an MCS that considers fault tolerance by design and answer the `RQ.S1`.
- In Section 2.3, we propose a distributed bio-inspired heuristic for **autonomous** service allocation and routing for any additional service demand without depending on a controller [ESF22]. The `JSAR` model is used for benchmarking to compare the proposed distributed heuristic with the optimal centralized solution. This contribution provides self-configuration procedures for service orchestration on top of the initial configuration and addresses the `RQ.S2`.
- In Section 2.4, we develop moving target defense (MTD) strategies by formulating an attacker-defender game for **security** of critical services [ESKF23]. The `JSAR` model is integrated into this game-theoretical model to find feasible service configurations to be deployed periodically within MTD strategies. This contribution eventually adds a protection layer above the initial service configuration, potentially (but not necessarily) leveraging the existing orchestration procedures, and addresses the `RQ.S3`.

After the preliminaries, we name the sections after their respective resilience goals, i.e., fault tolerance, autonomy, and security. Each section is then structured to present (i) a set of fine-grained research questions, (ii) our related publications, and (iii) the summary of contributions and their takeaways.

2.1 Preliminaries

In SOA, an overlay network of services is embedded into an underlying physical network to establish inter-service data traffic by fulfilling their latency and foremost resilience demands. A service overlay $O(S, D)$ consists of a set of services $s \in S$ and their communication demands $d \in D$. In a similar notation, a physical network $G(V, E)$ consists of nodes $v \in V$ and links $e \in E$. Here, a service s represents a function or virtual instance to be deployed on a physical node v. It has a certain level of criticality, e.g., mission-critical or best-effort, and consumes an amount of resources, τ_s, e.g., CPU or memory. Service criticality imposes a deployment constraint s.t. only particular nodes can host critical services, i.e., $k_{sv} = 1$, to establish their isolation and access control. Moreover, a demand d specifies inter-service communication requirements between two service instances in terms of the end-to-end latency l_d and the amount of data traffic h_d to be exchanged, e.g., required bandwidth[1].

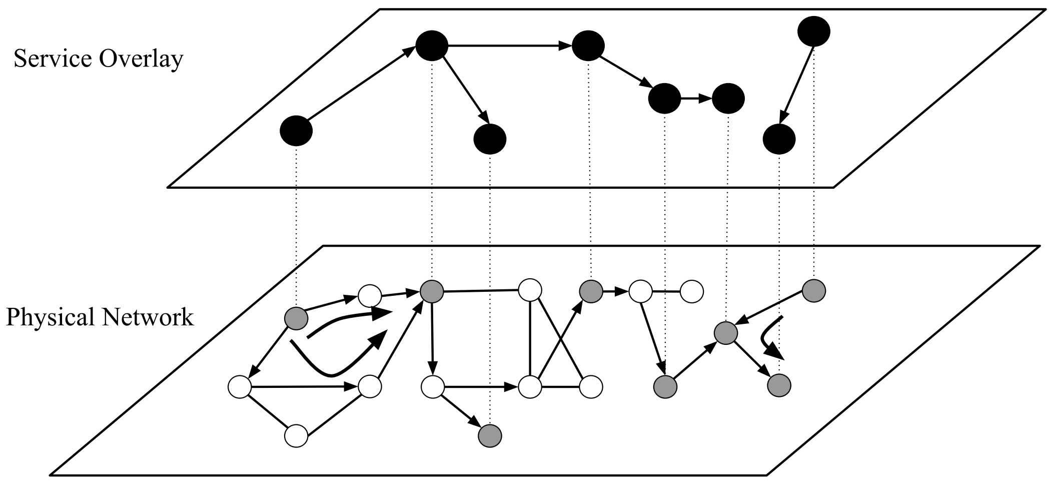


Figure 2.1: Service overlay on top of the underlay physical network. Dashed lines show how basic service instances are assigned to physical nodes. Grey nodes host the service instances, and directed edges show the paths carrying traffic demands.

Figure 2.1 gives an example of the embedding of a service overlay (black nodes) in the underlying physical network (grey nodes). While a link between two services (an edge between two black nodes) represents a demand, the connection of two physical nodes (an edge between two grey nodes) is a physical link eE, i.e., having a nominal bandwidth capacity, in the network. The overall deployment is restricted by (i) the resource capacities r_v of each node v and (ii) the bandwidth capacity

[1]The same notation is used for all the models in this chapter.

Table 2.1: Terms and definitions in the optimization problem. *Base* type contains the fundamental elements of the model. *Constant*s are network- and service-related parameters given as input. *Variable*s represent the parameters to be optimized.

Type	Symbols	Set	Interval	Definition
Base	u, v	V		Nodes in the network
	e	E		Link (edges) between nodes
	s, t	S		Basic services
	d	D		A demand between a pair of services
	p	P_{uv}		An end-to-end path between nodes u and v
Constant	τ_s	$\Re^*$	[0,∞]	Resource consumption of s
	h_d	$\Re^*$	[0,∞]	Traffic volume of d
	c_e	$\Re^*$	[0,∞]	Maximum link capacity of e
	r_v	$\Re^*$	[0,∞]	Maximum resource capacity of v
	n_s	Z^*	[0,∞]	Required number of instances for s
	l_d	$\Re^*$	[0,∞]	Latency requirement of d
	l_e^*	$\Re^*$	[0,∞]	Latency in e
	k_{sv}	Z^*	[0,1]	Binary variable to indicate if v is capable to host s
Variable	x_{dp}	$\Re^*$	[0,∞]	Flow allocated to path p of demand d
	y_{sv}	Z^*	[0,1]	Binary variable to decide if s is hosted by v

c_e of each link e. Besides, since each link induces a certain delay l_e, an end-to-end communication path $p \in P$ between any two nodes hosting connected services should be selected according to the latency requirement l_d between those services.

In the next part, we introduce the `JSAR` model that formulates the problem above as a mixed-integer linear programming (MILP) model. Table 2.1 summarizes all terms and definitions of the formulation. For the rest of this chapter, while the term JSAR *problem* refers to the overall joint service allocation and routing problem, the `JSAR` (only) refers to the particular MILP model that we propose.

Joint Service Allocation and Routing Model (JSAR)

As introduced above, in the JSAR problem, each service instance $s \in S$ should be deployed to one node to satisfy communication demands in the respective service overlay. A demand d is defined between a pair of service instances s.t. $\delta : D \mapsto (SxS)$ and $\delta_d = (s, t) \quad s, t \in S$ to regulate inter-service communication. Besides requiring an amount of data traffic between service instances, a demand also characterizes the QoS and resilience requirements for inter-service communication.

x_{dp} and y_{sv} are two binary decision variables to indicate if demand d is assigned to path p and if service s is deployed on node v, respectively.

$$\min \sum_{d \in D} \sum_{p \in P} x_{dp}|p| \tag{2.1}$$

$$\sum_{s \in S} y_{sv} \tau_s \leq r_v \qquad \forall v \in V \tag{2.2}$$

$$\sum_{v \in V} k_{sv} y_{sv} = 1 \qquad \forall s \in S \tag{2.3}$$

$$x_{dp} \leq y_{sv} y_{tu} + y_{tv} y_{su} \qquad \forall d \in D, \forall u, v \in V, \forall p \in P_{uv}, (s,t) \in d \tag{2.4}$$

$$\sum_{d \in D} \sum_{\substack{p \in P, \\ e \in p}} x_{dp} h_d \leq c_e \qquad \forall e \in E \tag{2.5}$$

$$\sum_{e \in p} x_{dp} l_e^* \leq l_d \qquad \forall d \in D, \forall p \in P \tag{2.6}$$

$$\sum_{p \in P} x_{dp} = 1 \qquad \forall d \in D \tag{2.7}$$

The (objective) Function 2.1 minimizes the length of selected paths, where $|p|$ represents the path length. Minimizing the total path length can be considered as both performance and cost optimization. That is, allocating shorter paths enables establishing low-latency communications, i.e., here with less hops, and decreasing the number of occupied links, which is especially important for mission-critical networks to reduce the cost and the complexity of the system.

Constraint 2.2 and 2.3 ensure that v has sufficient resources to host s and s is deployed on exactly one node that is capable to host s (e.g., equipped with the required hardware). Constraint 2.4 restricts the flow assignment in a way that d can be deployed on p if only the source and destination nodes u, v of p host required services s and t. Constraint 2.5 ensures that each link e of p has sufficient resources, e.g., bandwidth, to carry the traffic of d if it is assigned to p. While Constraint 2.6 ensures that p is selected to satisfy the maximum tolerable latency for d, Constraint 2.7 guarantees that d is assigned exactly to one working path. Note that, as inferred in the latest constraint, traffic demands are assumed to be non-bifurcated.

Note that the quadratic formulation in Constraint 2.4 is linearized using McCormick envelopes [Mcc76] introducing extra variables and constraints to make the model easily solvable by the existing linear optimization tools. Although the linearization details are omitted here, it is extensively discussed in [ERF20]. Referring to this linearized version, the `JSAR` is considered as a MILP in the rest of the thesis.

2.2 Fault Tolerance

This section introduces two fault tolerance extensions for the JSAR to answer the RQ.S1. The resulting models provide fault tolerance by reserving required resources for redundant service instances and rerouting in case of single node and link failures. The JSAR already provides an initial service allocation and routing scheme by optimally allocating sufficient node and link resources and satisfying different QoS requirements. However, critical services and data traffic in MCSs should also be resilient against potential failure scenarios by design. The state of the art (and the JSAR) does not holistically solve the joint service allocation and routing problem together with fault tolerance requirements, which can be significantly more complex than the JSAR problem. Besides, they do not reflect the interdependency of services in the mission-critical domain either. Therefore, the following questions arise in addition to the RQ.S1:

- RQ.S1.1: To what extent is it possible to protect service-oriented MCS against certain failure scenarios via dynamic service allocation and routing?
- RQ.S1.2: How much complexity and cost does the fault-tolerant SOA impose?

Accordingly, the following publication extends JSAR to reserve further resources for redundant service instances and their intercommunication optimally to failover in different failure states of a system[2]:

> Doğanalp Ergenç, J. Rak, M. Fischer. Service-Based Resilience for Embedded IoT Networks. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2020.

Beyond the initial configuration, reserving redundant resources for scaling and complex service overlays is substantially more challenging. Therefore, we extend the previous model further with a shared protection scheme, JSAR-SP, to share backup resources between different service and communication demands for resource efficiency instead of using dedicated backup capacity in the following publication[2]:

> Doğanalp Ergenç, J. Rak, M. Fischer. Service-based Resilience via Shared Protection in Mission-critical Embedded Networks. IEEE Transactions on Network and Service Management (TNSM), Special Issue on Design and Management of Reliable Communication Networks, 2021.

[2]In both publications, the whole contribution belongs to this thesis. The co-authors helped to improve the quality of the papers with their valuable feedback.

In the rest of this section, we present two extended models together with their key results in Section 2.2.1 and 2.2.2, respectively. The respective publications [ERF20] and [ERF21] are also attached in Appendix A and B.

2.2.1 Fault-tolerant Joint Service Allocation and Routing

In order to enhance the `JSAR` to find alternative service configurations against potential failures, we modify it by defining failure states. Failure state design is a concept where each state represents a failure scenario adding extra constraints to the optimization model [PM04]. It may include node failures, link failures, or both, and each state $f \in F$ is represented by additional input parameters, such as indices of failed nodes or links to characterize a failure. Only the initial state, i.e., $f = 0$, represents the natural state of a system without any failure. The model eventually provides alternative optimum deployments to be configured for each given failure scenario. A scenario might typically be an arbitrary node failure due to an attack, software failure, or power cut, affecting several service instances on the failed node(s). In each failure scenario, a set of binary parameters a_{vf} is given as input to specify if node v is available in failure state $f \in F$ and $\sum_{v \in V} a_{vf} = |N| - 1$ since only one node is assumed to be failed in each state.

In case of a failure, two main steps should be taken. First, all service instances hosted in the failed nodes must be deployed to other available nodes s.t.,

$$\sum_{v \in V} k_{sv} y_{svf} a_{vf} \geq 1 \qquad \forall s \in S, \ \forall f \in F \tag{2.8}$$

where y_{svf} represents if service s is deployed to node v in case of failure scenario f. Besides, if v hosts s in any f, there should be a reserved resource in v for s for migration in case that f occurs. Therefore, Constraint 2.2 is extended as,

$$\sum_{s \in S} \min\Big(\sum_{f \in F} y_{svf}, 1\Big) \tau_s \leq r_v \qquad \forall v \in V \tag{2.9}$$

Here, the term with min function indicates if s deployed to v in any number of states, only τ_s amount of resource needs to be occupied in v to activate that service.

Second, rerouting should be reconsidered since (a) paths may be broken due to failed nodes, and (b) the service deployment may change while migrating services in different failure scenarios. Thus, Constraint 2.4 is extended as,

$$\sum_{u,v \in V} a_{uf} a_{vf} \sum_{p \in P_{uv}} y_{suf} y_{tvf} \theta_{pf} x_{dp} \geq h_d$$
$$\forall d \in D, \ \delta_d = (s,t), \ s,t \in S, \ \forall f \in F \tag{2.10}$$

Table 2.2: The overview of the JSAR heuristics.

Heuristic	Greedy	Optimization	Service Deployment	Routing	Resilience	Scalability
RDDP	✓		✓	✓	$\sim$	✓
RDDP + BSRP	✓		✓	✓	✓	✓
MLSP	✓	✓	✓	✓		
MLSP + POBS	✓	✓	✓	✓	✓	

where $\theta_{pf} = \prod_{v \in V, v \in p} a_{vf}$ indicates if path p is available in state f, i.e., if all intermediate nodes in p are alive. We linearize this quadratic constraint to reduce the fault-tolerant JSAR to a MILP model with extra variables and constraints.

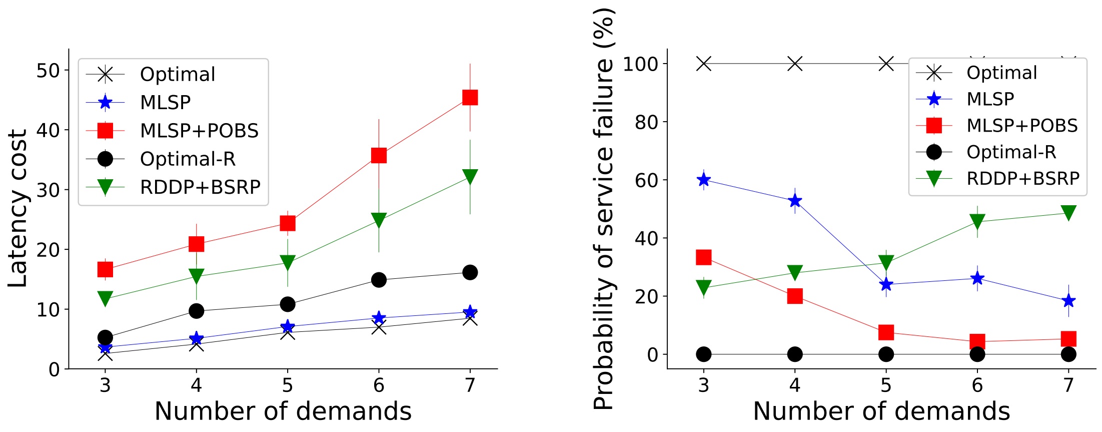
Note that the solution constructs a service deployment and routing scheme that is resilient to all single node failure scenarios. In this sense, such an approach can be considered both (i) protective, as it reserves required capacity in advance, and (ii) restorative, as it decides where to migrate services in case of related failure scenario happens.

Heuristics

Since the JSAR problem is known to be NP-hard, we additionally propose two heuristics, which cover all three dimensions of service deployment, inter-service traffic routing, and resilience. Table 2.2 summarizes the heuristics, including their approaches, i.e., greedy and optimized solutions, e.g., for service allocation, routing, and resilience.

The first heuristic consists of two greedy approaches to find an initial service deployment and then the placement of backup service instances. First, for each demand, Random Deployment with Disjoint Paths (RDDP) allocates services to a randomly selected pair of nodes by starting from the ones that do not host any other service. Then, two node-disjoint paths are allocated for the demand between selected nodes as working and backup paths. After finding a basis predeployment scheme with RDDP, Backups with Secondary Redundant Path (BSRP) selects a backup node for each service for the migration in case of a node failure. Similar to RDDP, BSRP deploys a backup service instance to the node with the highest available resources. As a result, RDDP+BSRP constitutes the complete greedy heuristic that handles service allocation, routing, and redundancy.

The second heuristic consists of three steps. Firstly, Maximum Load to Shortest Path (MLSP) places the most data-intensive demands, i.e., with high bandwidth requirements, to the shortest available paths and deploys related services to end-hosts of those paths, accordingly. Secondly, the rest of the demands, i.e., non-data-intensive ones, are again given to the reduced optimization problem under fault tolerance constraints. Then, Post-Optimization Backup Scheme (POBS) reserves suitable backup resources for the fixed services and demands that are priorly placed by MLSP in the



(a) Latency cost

(b) Probability of service failure

Figure 2.2: Impact of increasing demands for a topology with ten nodes.

first step by selecting the shortest backup paths between randomly selected nodes of backup service instances. Consequently, `MLSP+POBS` is the second heuristic, including both optimal and greedy solutions.

Further details of the heuristics can be found in the original paper [ERF20] (Appendix A).

Key Results

We evaluated fault-tolerant `JSAR` and the proposed heuristics in terms of latency cost ((objective) Function (2.1)) and probability of service failure (PoSF) for increasing number of services and demands. PoSF is the ratio of the number of services without backup nodes (due to insufficient node resources) to the number of all services. It represents the percentage of services that fail at most in case of an arbitrary node failure and is used as the fault tolerance metric. For the experiments, we used randomly generated topologies and service overlays. Our main goal for the evaluation was to achieve 100% fault tolerance against single random node failures with `JSAR` while satisfying all QoS requirements, and to reveal the computational overhead required for the optimal solution.

Figure 2.2 compares heuristics with the resilient *(Optimal-R)* (i.e., extended fault-tolerant `JSAR`) and non-resilient *(Optimal)* (i.e., the original `JSAR`). As shown in Figure 2.2a, the *Optimal* solution has the minimum latency cost, and `MLSP` gets close to the optimum. However, `MLSP` induces a PoSF of 20-60% as depicted in Figure 2.2b because of the pre-deployed services without backups. After *Optimal-R* (having 0% PoSF), `MLSP+POBS` offers the highest fault tolerance, which gets better from 40% to 5% with an increasing number of demands. However, in the `POBS` phase, it cannot find the optimum paths due to random search of the available nodes and becomes

2-4 times more costly compared to *Optimum-R*. While `RDDP+BSRP` is less costly than `MLSP+POBS` in terms of latency, it is negatively affected by the increasing demands and has a PoSF up to 50%. The reason is, increasing fault tolerance requires allocating more redundant resources, which might be reserved for active demands to assign shorter paths.

More comprehensive results on the model complexity, scalability, and parameter analysis can be found in the original paper [ERF20] (Appendix A).

Takeaways

- Although heuristics can find near-optimal solutions for JSAR, it is not possible to get similar results for fault-tolerant deployment, e.g., 0% PoSF, especially when the service overlay becomes more connected, i.e., with an increased service interdependency. Therefore, an accurate service overlay model should be defined depending on the domain requirements to avoid redundant computational effort and to obtain better (yet accurate) results. This also answers the `RQ.S1.1` by revealing the success of both the optimal and heuristics fault-tolerant solutions.
- Even for small service overlays, e.g., from 3 to 7 demands, it takes a long time with a decent computational cluster to find an optimal solution against all single node failures due to the significant complexity of `JSAR`. Therefore, further meta-heuristics are required for a better formulation. This takeaway also addresses the `RQ.S1.2` related to the complexity of the fault-tolerant design.

2.2.2 Resource Efficiency via Shared Backup Protection

As stated in the previous section, finding an optimal and fault-tolerant service deployment and routing configuration against different failure scenarios is highly complex. Therefore, we split that problem into three phases and develop different optimization models that need to be solved subsequently. In this split model, `JSAR-SP` (in which SP stands for shared protection), we employ a shared backup scheme instead of reserving dedicated resources for each backup path to reduce necessary backup capacity. By saving resources, it enables scaling an MCS with additional services and demands with potentially better QoS, e.g., shorter working paths. Accordingly, the overall goal is reaching the optimal solution more quickly for larger problem instances and also utilizing the existing resources more efficiently.

The three phases of the `JSAR-SP` is shown in Figure 2.3. In the bootstrapping phase, the `JSAR` finds an initial service deployment and the shortest working paths. In the second phase, shared backup protection, another MILP model `Backup-ILP` establishes shared backup paths against possible link failures on the working paths found in the first phase. It minimizes the use of backup resources by maximizing capacity sharing. In the service migration phase, the `JSAR` searches for the backup

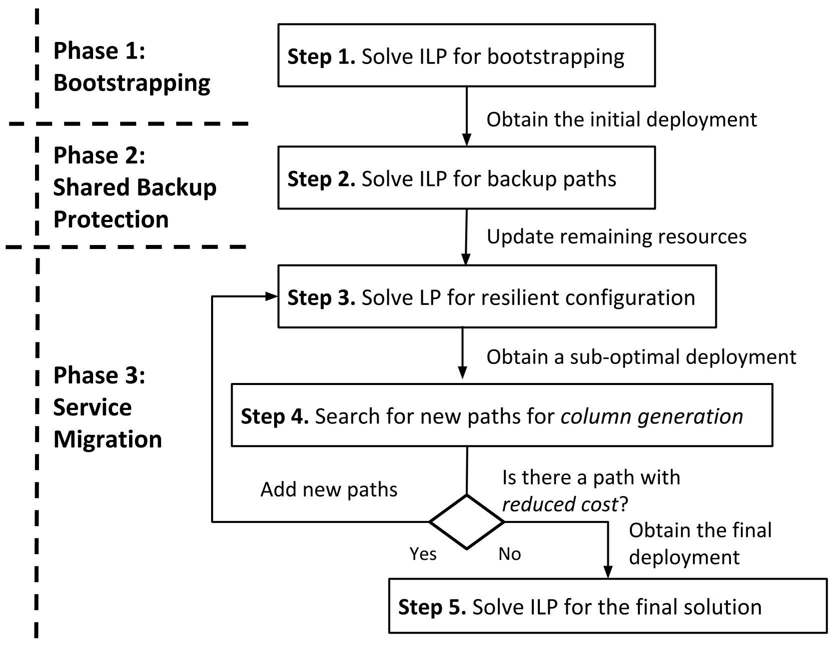

Phase 1: Bootstrapping
Step 1. Solve ILP for bootstrapping
Obtain the initial deployment
Phase 2: Shared Backup Protection
Step 2. Solve ILP for backup paths
Update remaining resources
Step 3. Solve LP for resilient configuration
Obtain a sub-optimal deployment
Phase 3: Service Migration
Step 4. Search for new paths for column generation
Add new paths
Is there a path with reduced cost?
Yes
No
Obtain the final deployment
Step 5. Solve ILP for the final solution

Figure 2.3: Three phases of the JSAR-SP.

nodes, which communicate with other host nodes with minimum latency, to migrate services in the case of node failures on a reduced problem set. Here, the JSAR is reformulated using column generation methods to solve larger problem instances. For the rest of this section, the shared backup protection phase is presented in more detail as the first, and the third phases are already variations of the JSAR. Besides, new heuristics are introduced to perform backup capacity sharing for scaling problem instances.

As presented in detail in [Rak12], for a demand d with the requested throughput h_d, the respective backup path at link e in the case of shared protection would require the allocation of (i) no extra capacity if the amount of shareable capacity c^+ already allocated to backup paths at link e is at least h_d or (ii) the extra capacity of $h_d - c^+$ in all other cases. The shareable capacity can be considered as the capacity already reserved for a backup path of another demand q that is accepted earlier and not affected by the same link failure affecting a working path of d. In case $h_q < h_d$, extra capacity $h_d - h_q$ needs to be reserved at the link even though h_q amount of capacity can be used by both q and d in case of (different) link failures.

Accordingly, the Backup-ILP is given below. Using the initial configuration (i.e., working paths) as the input, a set H_d is constructed for each d. It includes demands $\{q_1, q_2...\}$ that (i) induce shareable backup capacity with demand d as they have disjoint working paths with d and (ii) have larger traffic demands $h_q > h_d$. As a result, if $q \in H_d$ is assigned on link e with the available capacity c_e^*, h_d is not necessary to consume extra capacity. The binary decision variables z_{dp} and z_{de}^* represent whether demand d is assigned to path $p \in P_{\bar{d}}$ and link e, respectively.

Here, $P_{\bar{d}}$ is a set of disjoint paths to the working path of d obtained from the previous phase, and it is computed beforehand. The other decision variable g_{de} shows if any $q \in H_d$ is already assigned to link e. $\bar{g_{de}}$ is the negation of g_{de}.

$$\min \sum_{e \in E} \sum_{d \in D} z^*_{de} \bar{g_{de}} h_d \tag{2.11}$$

$$\sum_{p \in P_{\bar{d}}} z_{dp} = 1 \qquad \forall d \in D \tag{2.12}$$

$$z^*_{de} \geq z_{dp} \qquad \forall d \in D, \forall e \in E, p \in P_{\bar{d}} \land e \in p \tag{2.13}$$

$$z^*_{de} \leq \sum_{\substack{p \in P_{\bar{d}}, \\ e \in p}} z_{dp} \qquad \forall d \in D, \forall e \in E \tag{2.14}$$

$$g_{de} \geq z^*_{qe} \qquad \forall d, q \in D, \forall m \in H_d, \forall e \in E \tag{2.15}$$

$$g_{de} \leq \sum_{q \in H_d} z^*_{qe} \qquad \forall d \in D, \forall e \in E \tag{2.16}$$

$$\sum_{d \in D} z^*_{de} \bar{g_{de}} h_d \leq c^*_e \qquad \forall e \in E \tag{2.17}$$

In the model, (objective) Function 2.11 minimizes the total backup resources by enforcing capacity sharing. It eventually increases resource efficiency and decreases the design cost of the system. Constraint 2.12 ensures exactly one backup path assigned for each demand d. Constraints 2.13 and 2.14 configure z^*_{de} for each link e checking if p involving e is a backup path for demand d, i.e., $z_{dp} = 1$. Similarly, Constraint 2.15 and 2.16 configure g_{de} checking if any demand $q \in H_d$ is assigned to link e. Lastly, Constraint 2.17 ensures that the required resources for d with the highest traffic demand are reserved.

Heuristics

The three phases of `JSAR-SP` essentially handle (i) service deployment satisfying demands, (ii) shared link protection scheme minimizing the use of link resources, and (iii) service migration scheme for node protection. We also propose a 5-step heuristic to cover these phases with significantly lower computational complexity. Figure 2.4 shows the corresponding steps. Step 1 (Service deployment) and step 3 (Finding working paths) correspond to the bootstrapping phase, which performs initial service deployment and assignments of shortest working paths. In step 2 (Shared link protection), necessary backup paths are found for each demand in a way to maximize the shared use of links similar to the shared protection phase. Note that instead of finding working paths in advance, the heuristics perform a shared link protection scheme before assigning working paths, i.e., step 3 before step 2. The reason is that the assignment of working paths restricts the available links to be used in backup paths significantly, as they should be disjoint not to be affected by the same single link failure (and be eligible for capacity sharing). However, after reducing the number of used links and required backup capacity by sharing, there are still sufficient resources left to establish shorter working paths later. Step 4 (Assigning remaining demands) plays a complementary role to steps 1-3 to ensure that the working and backup paths are assigned for all demands with minimal path lengths and maximal backup capacity sharing. Lastly, step 5 (Finding backup nodes) is to find alternative service configurations utilizing the shortest available paths to set up in the case of failures, corresponding to the service migration phase. In the rest of this part, steps 2 and 3 are explained in more detail as they directly correspond to the `Backup-ILP` model introduced at the beginning of this section. The other steps reuse several heuristics presented in Section 2.2.1.

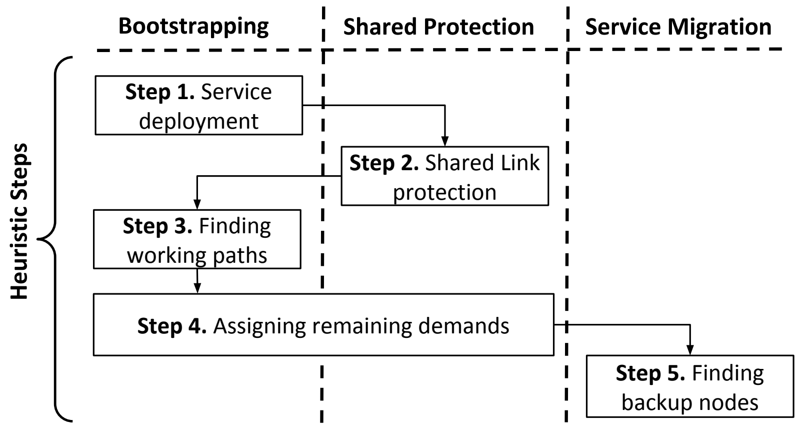

Bootstrapping
Shared Protection
Service Migration
Heuristic Steps
Step 1. Service deployment
Step 2. Shared Link protection
Step 3. Finding working paths
Step 4. Assigning remaining demands
Step 5. Finding backup nodes

Figure 2.4: The correspondence between the heuristics and the `JSAR-SP`.

A service hosted at a particular node can receive and send data to different services on other nodes to satisfy various communication demands. This interdependency leads to a chain of services. When such services are allocated at physical nodes, it is convenient to define a communication backbone connecting and covering all those nodes to be shared by multiple demands in case of failures. Accordingly, in step 2, Secondary Backup Backbone (`SBB`) algorithm distinguishes the chain of services whose host nodes can also be connected sequentially to form a backup backbone. That is, the service chain in the service overlay is also reflected as a chain of physical nodes forming a single backbone. For each service chain, `SBB` constructs a modified Steiner tree [MP92] on the physical network. A Steiner tree is defined as a connected subgraph including a set of *selected* nodes, e.g., terminal nodes, belonging to a graph. `SBB` constructs a secondary Steiner tree, which selects the shortest disjoint path to the shortest path between two terminal nodes so that the shortest paths can be (potentially) used as working paths in the next step. As a result, this backup backbone can be utilized by all services hosted at the terminal nodes when their respective paths fail.

In the third step, the Mutually Disjoint Paths (`MDP`) algorithm calculates the working paths. They should be mutually disjoint if the backup paths of the respective demands are shared. Even though there is a single backup backbone constructed in the previous step, each demand $d \in D$ uses only a segment of the backbone G_{Steiner}, i.e., a path between the nodes hosting the services of that demand, that could be shared or not. Therefore, when finding a working path, it is first checked if d utilizes any capacity shared with another demand $q \in H_d$ with an assigned working path w_q and ensure that they are disjoint. Another important issue is, that the working path of d, p_d^w should be disjoint to the respective segment of the backup backbone p_d^b used for that demand. Those two steps eventually construct the fault-tolerant communication scheme employing capacity sharing. Further details of the heuristics can be found in the original paper [ERF21] (Appendix B).

Key Results

We evaluated the `JSAR-SP` and the heuristics in terms of probability of service failure (PoSF) and sharing efficiency. The PoSF is the same measure used in Section 2.2.1, and the sharing efficiency represents the capacity gain by sharing. It is the ratio of the difference between shared and dedicated backup capacity to the dedicated backup capacity without sharing. Our main goal in this evaluation is to show that saving backup capacity with `JSAR-SP` enables more available resources and allocating more services and demands while still achieving 100% fault tolerance against single node failures.

We designed a realistic in-plane network topology with isolated aircraft partitions, e.g., node, tail, wings, and cabin, and restricts the service placement according to the nodes' capabilities in different partitions. Besides, the generated service overlays reflect well-defined service characteristics, including data demands and criticality. For benchmarking, random topologies and service overlays are used as well. In the figures below, while *Optimal* and *Heuristic* values represent the optimal and the heuristic's results for the given in-plane topology, *Optimal-R* and *Heuristic-R/* show the results for random networks of the same size.

Figure 2.5 shows the PoSF in case of a single node failure for increasing communication demands. As the optimal solution guarantees finding a backup path for each demand and a backup node for each service, it protects the network against all potential single node and link failures. Therefore, the optimal deployments at both in-plane and random topology result in 0% PoSF. Similarly, the heuristic is also successful in reserving the required capacity in backup paths for all the demands. However, in the last stage of the heuristic, there are some scenarios where it fails to find alternative nodes to migrate services due to (i) insufficient amount of node resource capacity, (ii) link capacity, or (iii) lack of suitable paths satisfying the latency requirements. For the in-plane topology and random networks, our heuristic keeps the PoSF below 5% and 10%, respectively.

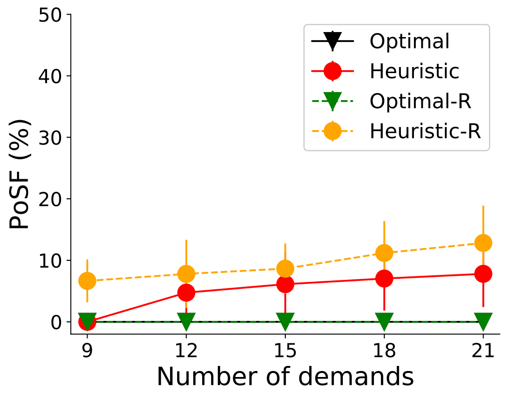


Figure 2.5: Probability of service failure in case of a single node failure.

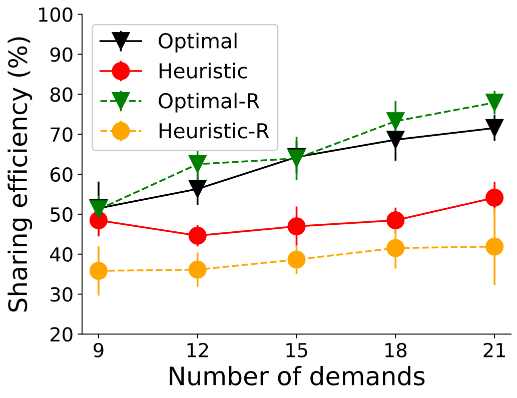


Figure 2.6: Sharing efficiency.

Figure 2.6 shows the sharing efficiency, which is the gain of using the shared capacity instead of dedicated capacity for increasing communication demands. While `Backup-ILP` can utilize backup paths to decrease the required backup capacity by 50-75%, the heuristic gives steady results around 50% and 40% backup capacity savings for in-plane (*Heuristic*) and random (*Heuristic-R*) topologies.

Further analysis of capacity sharing and scalability, and the details of in-plane topology and service overlays can be found on the original paper [ERF21] (Appendix B).

Takeaways

- An efficient use of resources by capacity sharing leads to a better service deployment that leaves sufficient resources for backup service instances and paths. In comparison to the results of the heuristics in Section 2.2.1, we obtain 4-5 times less PoSF for larger problem instances. It eventually indicates better fault tolerance with less computational effort and answers the `RQ.S1.1`.
- By rearranging backup paths, it is possible to save more than 70% of backup resources, which substantially impacts the design cost and resource utilization. This also addresses the `RQ.S1.2` in terms of the cost of optimal and fault-tolerant service-oriented design.

2.3 Autonomy

—via Bio-inspired and Distributed Service Orchestration

This section introduces a distributed and bio-inspired service orchestration framework to solve the JSAR problem and to answer the `RQ.S2`. It minimizes the dependency of MCSs on a controller and eventually provides autonomy for the service configuration.

Currently, in different areas such as SDN and cloud computing, service allocation, and routing problems are handled by a controller with network-wide visibility [MSRL+14]. Similarly, our first proposal `JSAR` can be solved centrally by a computationally capable entity, i.e., by a dedicated controller or server. However, centralized approaches usually have limited scalability and introduce a single point of failure and thus safety risks for MCSs. Therefore, decentralized and distributed orchestration techniques are required.

For distributed and self-organized task allocation, bio-inspired algorithms (BIAs) are employed in various networking domains [LD07, DA10]. They adapt the colony behavior of different animal species, such as ants and bees that collaborate in nature via specific communication patterns to solve domain-specific problems. However, those existing works on distributed service management do not reflect the characteristics of mixed-criticality and interdependent services of MCSs. Besides, it is usually not possible to justify their performance and complexity compared to the centralized solutions. Therefore, the following questions arise in addition to the `RQ.S2`:

- `RQ.S2.1`: How efficient is a self-organized service orchestration framework compared to the traditional centralized approach?
- `RQ.S2.2`: What is the additional complexity and cost of a distributed service allocation and routing solution?

Accordingly, the following publication introduces a bio-inspired ant colony optimization (ACO) framework to solve the JSAR problem distributedly with minimum dependency on a controller and compares it with `JSAR`[3]:

```
Doğanalp Ergenç, D. Sorejevic, M. Fischer. Distributed Bio-inspired
Configuration of Virtualized Mission-critical Networks. IEEE Global
Communications Conference (GLOBECOM), 2022.
```

[3]For the given publication, the main contribution belongs to this thesis. The second co-author evaluated an early version of the proposed framework in the context of his master's thesis. However, it is here remodeled from scratch and extended with an optimization model. The evaluation is also re-conducted with new metrics. The third co-author helped to improve the quality of the paper with his valuable feedback.

In the rest of this section, we present the details for adaptation of ACO to the JSAR problem and the results of this model to answer the mentioned questions. The respective publication [ESF22] is also attached in Appendix C.

Bio-inspired and Distributed Service Orchestration

Our bio-inspired framework performs service allocation and routing consecutively together with a redeployment phase in three stages.

1. First, all nodes are initialized with the information on available services and inter-service communication demands. They start the *service allocation* stage by selecting a set of services probabilistically within their available resources utilizing ACO functions[4]. They then broadcast their selections and obtain some of them after reaching a consensus in multiple iterations.

2. Secondly, according to the given deployment, they initiate the *path discovery and routing* stage leveraging a modified version of the distance-vector routing algorithm and the adaptation of the ACO functions. Note that each node utilizes two ACO functions for these stages: (i) the first function calculates the probability of deploying a particular service, i.e., service probability, and (ii) the second function is to select a forwarding link probabilistically for a particular demand.

3. When service allocation and routing are handled separately in sequence, a service deployment may render inter-service routing infeasible due to the violated QoS constraints or a lack of sufficient capacity on the particular paths. In the *redeployment* stage, the respective services of the violated demands are migrated to different nodes until finding feasible paths for their intercommunication.

In the formulation of each stage, we consider the safety requirements of MCSs by (i) avoiding the deployment of several critical services to the same node physical node and (ii) load-balancing the traffic load to widen the demand assignment instead of relying on the dedicated shortest paths. Both prevent a system from a single point of failure and performance bottlenecks, which may cause disruptions in critical services and traffic.

In the rest of this section, we explain the different adaptations of ACO functions to the service allocation and routing stages together and present two alternative redeployment strategies.

[4]For the details of the ACO framework, interested readers can refer to the seminal paper by Dorigo [DG97].

Service allocation

The service allocation stage imposes two constraints: A node can host (i) a limited number of services within its resource capacity and (ii) a limited number of critical services to avoid a single point of failure. Initially, each node obtains a list of services S and demands D from a controller unit, which does not maintain or control the system later on. Then, every node $u \in V$ calculates a probability for each service $s \in S$ to decide on the set of services that they can host. The respective probability function $P_u(s)$ is given in Equation 2.18.

$$P_u(s) = \frac{\mu_u(s)^\beta}{\sum_{t \in S} \mu_u(t)^\beta} \tag{2.18}$$

Here, $\beta \in [0,1]$ is the ACO learning parameter that adjusts the exploration behavior of the ACO heuristic and increases the deployment probability of a service. The threshold function $\mu_u(s)$ of ACO is calculated as

$$\mu_u(s) = \begin{cases} 0, & \text{if } r_u^* < \tau_s \\ \max\{r_v - \sum_{t \in S_u} y_{tu}(\lambda o_t \tau_t + (1 - o_t)\tau_t), 0\} \end{cases} \tag{2.19}$$

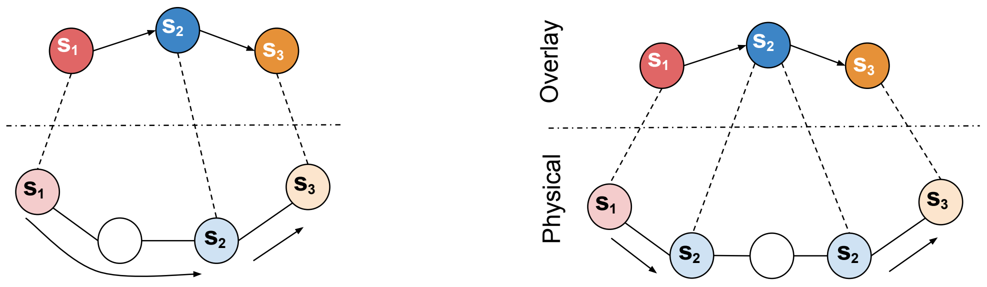
λ in Equation 2.19 represents the weighting factor for the critical services, i.e., $o_s = 1$. It penalizes the deployment of an excessive number of services proportional to their resource consumption. Each node then calculates a deployment probability starting from the critical services. The increasing resource consumption here is reversely proportional to the increasing pheromone level in the original ACO framework [DG97].

After deciding on the services to host, all nodes broadcast their decisions together with the computed result of $P_u(s)$. When there is more than one candidate node for the same service, the one with the highest probability gets the respective service. Meanwhile, nodes also identify the remaining unclaimed services. Then, for the following iterations of the service allocation, they announce which of the remaining services they can host additionally by calculating probabilities similar to the first iteration.

Routing

In the routing stage, each node u utilizes two distance functions $D_u(v)$ and $D_u^i(v)$ to calculate the probability of forwarding a traffic demand d from to the destination node v via next-hop i. While $D_u(v)$ is the cost of the shortest path between u and v, $D_u^i(v)$ is the cost of the path passing over i and destined to v. Accordingly, Equation 2.20 computes the given probability as

$$P_{uv}^i(d) = \frac{r_{uv}^i(d)^\alpha}{\sum_{j \in N_u} r_{uv}^j(d)^\alpha} \tag{2.20}$$



(a) Strict (b) Flexible

Figure 2.7: Different redeployment strategies.

where α is the ACO learning parameter. Moreover, $r^i_{uv}(d)$ is the ACO threshold function calculated as

$$r^i_{uv}(d) = \begin{cases} 0, & \text{if } D^i_u(v) > l_d \text{ or } c^*_e < h_d \\ \frac{\gamma_1 D_u(v)}{D^i_u(v)} + \frac{\gamma_2 c^*_e}{c_e}, & \text{otherwise} \end{cases} \tag{2.21}$$

where c^*_e represents the available resources on link $e \in (u, v)$ and $\gamma_{1,2}$ are the weights s.t. $\gamma_1 + \gamma_2 = 1.0$ to adjust the impact of the cost and resource utilization. Note that the link with higher available bandwidth and leading to the path with a lower cost has a higher probability according to Equation 2.21. Therefore, the link utilization is reversely proportional to the pheromone level in ACO.

Redeployment

After the initial service allocation, some demands may not be satisfied since (i) the distance, e.g., in terms of the number of hops, between the nodes u and v hosting respective services s and t might be too far to satisfy the required QoS or (ii) there might not be sufficient bandwidth on any available path between those two nodes. In such cases, our framework follows one of the following redeployment strategies by (i) deploying duplicate service instances to satisfy the remaining demands or (ii) migrating the respective services of unsatisfied demands and embedding the service overlay as it is. They are also illustrated in Figure 2.7.

Strict redistribution: Figure 2.7a illustrates the first redeployment strategy, `bia-strict`. It deploys the service overlay as a whole, assuming that each service *strictly* requires the data from the preceding service to process and send to the successor service. In the figure, s_2 is deployed on a node between the hosts of s_1 and s_3 to receive data from s_1, process them, and forward to the s_3 without losing any contextual information contained in the input of s_1. If the requirements of even one of the corresponding demands cannot be satisfied, `bia-strict` migrates the service instances *starting from the services that are least common among deployed demands*. In this scenario, s_2 could be moved until both demands between $s_1 - s_2$ and $s_2 - s_3$ are satisfied simultaneously. Eventually, `bia-strict` triggers several iterations of redeployment until all demands are assigned. To minimize its convergence time, `bia-strict` also penalizes the nodes that fail to host the selected services after each iteration to avoid the same unsuccessful service allocation. We also propose another redeployment strategy that trades off extra resource use for a better convergence time.

Extra service deployment: If the services are not strictly interdependent, it is possible to deploy two instances of s_2 to relax the service allocation constraints s.t. satisfying $s_1 - s_2$ and $s_2 - s_3$ separately instead of forming a chain of services. For such cases, the redeployment strategy `bia-flex` deploys duplicate service instances *flexibly* to satisfy the different demands that utilize the same services. It accordingly allows multiple nodes to host an instance for any remaining service. Consequently, both demands between $s_1 - s_2$ and $s_2 - s_3$ can be assigned to the shortest paths possible without potentially violating any QoS requirements, e.g., they induce only 1-hop latency in this scenario.

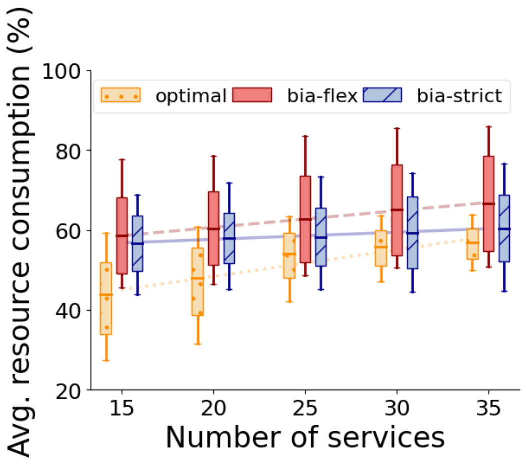
`bia-flex` loosens the assumption that services in the given overlay are tightly dependent on each in a given order and extends the solution space. However, it also costs extra node resources to host duplicate service instances.

Key Results

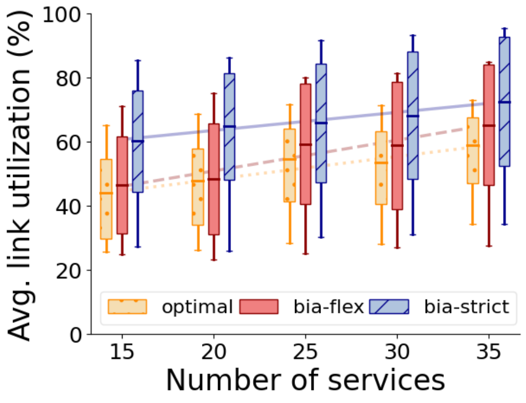
We evaluated our bio-inspired service orchestration framework regarding the fairness in resource utilization and the redeployment cost. We also compared the individual results with the results of the centralized and optimal solution, which is obtained from the `JSAR` by assuming that it is deployed in a centralized controller. For that, the original objective function (2.1) is replaced with a multi-objective function (2.22) that ensures fairness by minimizing the maximum link utilization and node resource consumption and is given below.

$$\min\Bigg\{ \max\Big\{\epsilon_l \sum_{d\in D} \sum_{\substack{p\in P,\\ e\in p}} x_{dp}h_d : \forall e \in E\Big\} + \max\Big\{\epsilon_n \sum_{s\in S} y_{su}\tau_s(1+o_s(\lambda-1)) : \forall u \in V\Big\}\Bigg\} \tag{2.22}$$

Such an objective (i) balances the resource use, (ii) promotes further configurability in case of continuous deployment of new services and demands, and (iii) reduces the risk of a single point of failure that may happen many services and demands are deployed on the same network elements. Our main goal in these experiments is to achieve near-optimal results with our distributed framework with a negligible control overhead.



(a) Node resource consumption



(b) Link utilization

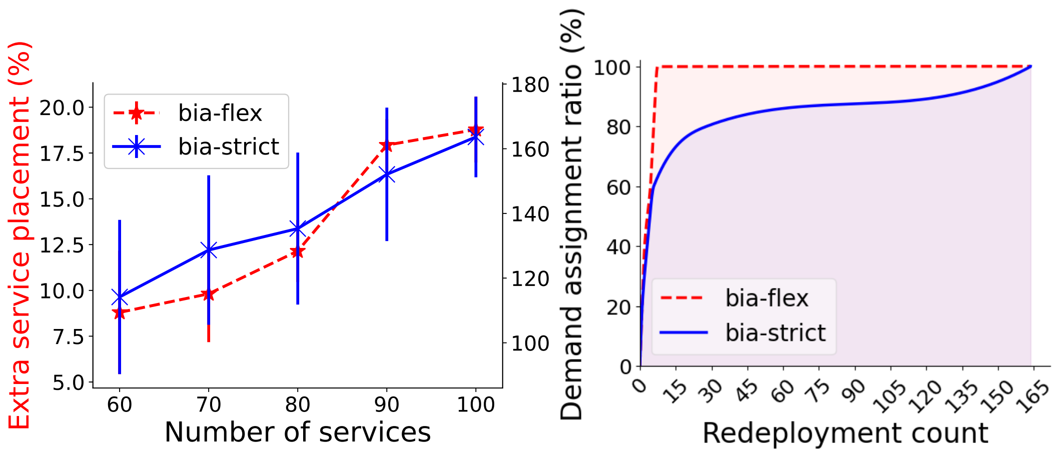
Figure 2.8: Resource utilization for increasing size of service overlay.

In all experiments, our framework successfully embedded all the services and demands with varying resource utilization and control overhead depending on the redeployment strategy. Figure 2.8 shows the node resource utilization and link utilization, respectively. The box plots for *optimal* (orange, dotted), `bia-flex` (red, solid), and `bia-strict` (blue, hatched) show the (averaged) minimum, maximum (edges of the vertical lines), and mean (mid-line) utilization of the active network components, and the standard deviation (boxes around the mid-line) in resource consumption. While a smaller maximum value and standard deviation indicate a fairer resource usage, the mean utilization shows resource efficiency when increasing the service overlay size.

In Figure 2.8a, `bia-flex` leads to a higher utilization of node resources than `bia-strict` because it requires placing extra service instances. The difference in the maximum utilization stems from that the nodes with the highest connectivity, i.e., the most central ones, tend to host those instances more often. The reason is, that after multiple redeployment iterations, most of the nodes still cannot satisfy the QoS requirements of the remaining demands, and such high-connectivity nodes provide the most suitable routes. In contrast, `bia-strict` stays closer to the *optimal* regarding the mean utilization.

Figure 2.8b shows that `bia-strict` results in the highest maximum link utilization and thus causes more link congestion. It leads to a denser service and data traffic deployment, as all service instances should be connected strictly. In contrast, `bia-flex` utilizes a broader range of different links by initiating extra service instances on different nodes. Therefore, it enables the use of alternative routes. Accordingly, for the link utilization, `bia-flex` gives closer results to the optimal solution.

Figure 2.9 shows the redeployment cost of our framework in terms of the extra service placement and the redeployment (convergence) time. In Figure 2.9a, the



(a) Cost of redeployment strategies (b) Demand assignment distribution

Figure 2.9: Replayment cost in terms of extra services and iterations.

left y-axis (red) shows the percentage of the services that require extra instances for `bia-flex`, and the right y-axis (blue) shows the required number of redeployment iterations for `bia-strict`. Note that `bia-flex` also requires a negligible number of redeployment iterations, e.g., less than 10 in the given scenarios. As seen in the figure, while `bia-flex` can require up to 20% extra instances for the largest service overlay, `bia-strict` takes more than 150 redeployment iterations, which induces further convergence time and also data overhead for the consensus between the nodes. Therefore, while the former costs extra node resources, the latter requires a longer time to settle the service configuration.

Although `bia-strict` requires a high number of redeployment iterations for a fully-functional network, both heuristics deploy the majority of the demands rather quickly. Figure 2.9b shows the cumulative percentage of the demand assignment by the number of redeployment iterations. While `bia-flex` (red, dashed) deploys all services and demands under 10 iterations by placing extra service instances, `bia-strict` can assign 80% of the demands in the first 30 iterations. The reason is that the most connected services, i.e., the services utilized by many demands, should be replaced until they satisfy the QoS requirements of all demands. More results on scalability and QoS can be found in the original paper [ESF22] (Appendix C).

Takeaways

- Depending on the redeployment strategy, it is possible to obtain closer results to the optimal in terms of node and link resource utilization, while the `JSAR` with multi-objective function can optimize both at once. However, our framework is significantly faster than `JSAR`, which corresponds to a centralized solution. This shows the efficiency of distributed and bio-inspired framework in comparison to the centralized and optimal solution, and addresses the `RQ.S2.1`.

- It is possible to use `bia-flex` for dynamic or incremental service and demand assignment as it has more than 10 times better convergence time with up to 20% extra resource cost. Both can also be used together to first deploy a fully-functional network with all demands in place using `bia-flex` and then to shift to `bia-strict` for incoming service demands to have better resource efficiency. This answers the `RQ.S2.1` related to the cost of a distributed solution.

2.4 Security

—via Moving Target Defense for Service-oriented MCSs

Service-oriented architecture provides additional flexibility to MCSs to withstand and mitigate attacks, e.g., by migrating critical services and data flows. Accordingly, this section introduces a spatio-temporal model that reconfigures the services over time by utilizing `JSAR` within moving target defense (MTD) strategies to answer the `RQ.S3`. These strategies specify when and how to reconfigure a system to strip attackers from their asymmetric advantage from long-term reconnaissance of MCSs and prevent targeted attacks, e.g., advanced persistent threats (APTs).

However, MTD via service reconfiguration requires additional spare resources and induces further reconfiguration overhead in terms of an increased delay and potential service interruptions to migrate or reinitiate the services. Another challenge is, a single MTD strategy should be effective against many likely attack scenarios. Therefore, the following questions arise in addition to the `RQ.S3`:

- `RQ.S3.1`: How (often) should services be reconfigured to prevent calibrated attacks and block persistent threats without interrupting service availability?
- `RQ.S3.2`: How effective is dynamic service (re-)configuration in hindering attackers' reconnaissance efforts and persistent threats?

Several works have revealed theoretical bounds for the cost of MTD strategies or propose MTD solutions only for singular and domain-specific attacks. In contrast, the following publication introduces a more realistic model for service-oriented MCSs, combining `JSAR` with an attacker-defender game to optimally reconfigure the service deployment over time[5]:

Doğanalp Ergenç, F. Schneider, P. Kling, M. Fischer. Moving Target Defense for Service-oriented Mission-critical Networks. International Conference on Computer Communications and Networks (ICCCN), 2023.

In the remainder of this section, we first explain the basics of the attacker-defender game. It is then formulated as a MILP model, `PLSCH`, to provide an optimal reconfiguration schedule against various attack scenarios. Then, we combine `JSAR` and `PLSCH` to find feasible service configurations within this schedule. The respective publication [ESKF23] is attached in Appendix D.

[5]In the forementioned publication, the main contribution belongs to this thesis. The second co-author contributed to writing preliminaries explaining an existing game-theoretical model. All co-authors also helped to improve the quality of the paper with their valuable feedback.

Moving Target Defense for Service-oriented MCSs

MTD is an umbrella term that covers dynamic reconfiguration and adaptation strategies for various systems to defend them against their respective security threats. They should be developed regarding the critical assets of the target system and the characteristics of its threats. In this thesis, MTD strategies are formulated within an attacker-defender game, Probabilistic Learning Attacker and Dynamic Defender (PLADD), which was originally proposed by Jones et al. [JOG+15, JOG+17]. The PLADD model captures the varying timing characteristics of the attacks, especially relevant to critical networks, which are exposed to various threats, from relatively fast reconnaissance attempts to well-calibrated attacks. In the context of SOA, we consider *services*, especially the safety-critical ones, as the most critical assets to be protected with MTD strategies.

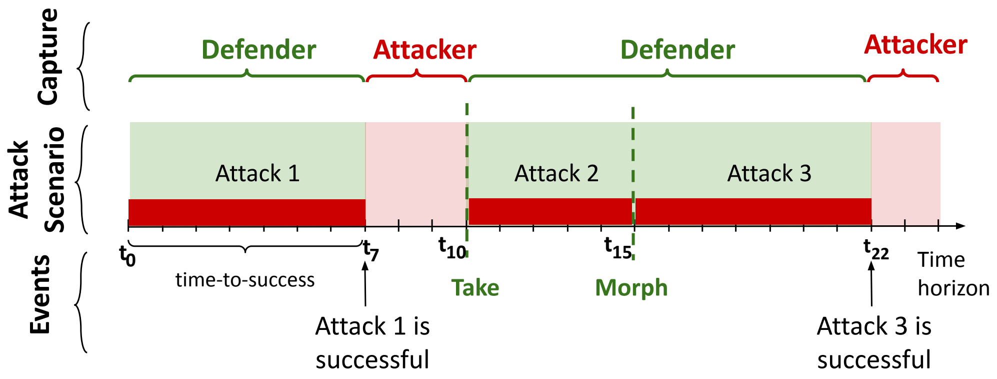


Figure 2.10: An example of the PLADD game.

Figure 2.10 shows the basics of the PLADD game. It involves (i) an attacker with learning capabilities and (ii) a defender capable of two different actions competing to gain control of the system within a given *time horizon*, i.e., the system's operational time. An attacker can conduct successive attacks (red blocks) that each take a certain time to be completed, i.e., having *time-to-success*. As a result of a successful attack, the attacker takes control of the respective services or resources (light red background from t_7 to t_{10}). When an attacker completes an attack, it might also *learn* about the system, and its subsequent attack takes less time, e.g., attack 2 is shorter than attack 1.

The role of a defender is to conduct defensive actions (vertical dashed lines) to prevent an attacker from completing its attack and to keep control over the system. A *take* action usually represents an instant intervention, e.g., resetting a service instance, while a *morph* represents substantial system changes, e.g., migration of multiple service instances among the nodes with diverse configurations and rerouting their inter-communication. After each action, the defender regains control over the system in case it is compromised by the attacker. Additionally, after a morph action, the attacker loses its insights on the system, i.e., what it learned from the previous attacks, and thus has a longer time-to-success for its following attacks. Therefore, the attacker immediately starts a new attack after each action. Accordingly, depending on the potential attack scenarios towards the target system, the defender should implement a schedule of actions so that even though it cannot directly detect any compromised assets or malicious attempts, it can block persisting attacks against the critical parts of the system. As a restriction, it cannot take those actions too often since they may cause service interruptions and may hinder the strict availability requirements of MCSs. Therefore, the defender has a limited *budget* to conduct its defensive actions.

Eventually, the overall goal of the PLADD game is to find a schedule of sequential defensive actions against potential security threats with certain timing characteristics, considering the capabilities and limitations of both attacker and defender sides.

PLADD Scheduling (PLSCH)

We formulated the forementioned PLADD as a MILP model, PLADD-Scheduling (PLSCH), to obtain concrete and optimal time schedules for defensive actions. This formulation also renders its combination with the JSAR easier, whose details we present in the following subsection. Originally proposed in [PPP+18], PLSCH represents the attacker-defender game as a combinatorial job assignment problem. It first considers a system of $m \in \mathbb{N}$ machines over a time horizon of $T \in \mathbb{R}^+$ time units. Each machine m comes with a job sequence $J_m = (d_{m1}, d_{i2}, \dots)$ of at most $n + 1$ jobs which it must process. The *duration* $d_{ij} > 0$ of the jth job on machine m specifies how long it takes machine m to process its jth job. In order to start processing the jth job of J_m, machine m must have finished the first $j - 1$ jobs. Additionally, the starting time of jobs is restricted to times $t \in [0, T)$ at which one of up to n many *takes* has been scheduled. So if a take is scheduled at time t (and only then), any machine m idle (i.e., currently not processing a job) may start to process its next job from J_m.

In the job assignment problem, each machine m represents an attack scenario, and each job j represents an attack as a part of a particular attack scenario. Here, the order of jobs on a machine indicates different, intentional, and organized attack steps of, for instance, a well-calibrated and persistent attack. A finishing job after d_{mj} amount of time means a completed attack after which the attacker captures control of (possibly certain parts of) the respective system. It is assumed that the starting time of a job corresponds to when the defender takes action. This assumption infers that the attacker starts to conduct a new attack as soon as the defender takes action. Moreover, any time difference between the end of a job and the beginning of the next job represents the time that the attacker captures control over the system. With this interpretation, the objective in PLSCH becomes to schedule the maximum number of jobs over all machines during $[0, T]$ so that such an idle time between any two jobs is minimized, i.e., minimizing the attacker's capture time.

Note that the defender does not know which particular attack scenario could occur. Instead, it aims to have a single schedule of defensive actions that could protect the system against many potential attack scenarios at once. Figure 2.11 illustrates this difficulty. While there is no idle time on the first machine (attack scenario 1), the same schedule results in more idle times on the other machines (attack scenarios 2 and 3). It means that even though this MTD schedule would be effective against the first attack scenario, the second and third scenarios would result in a higher attacker's capture time.

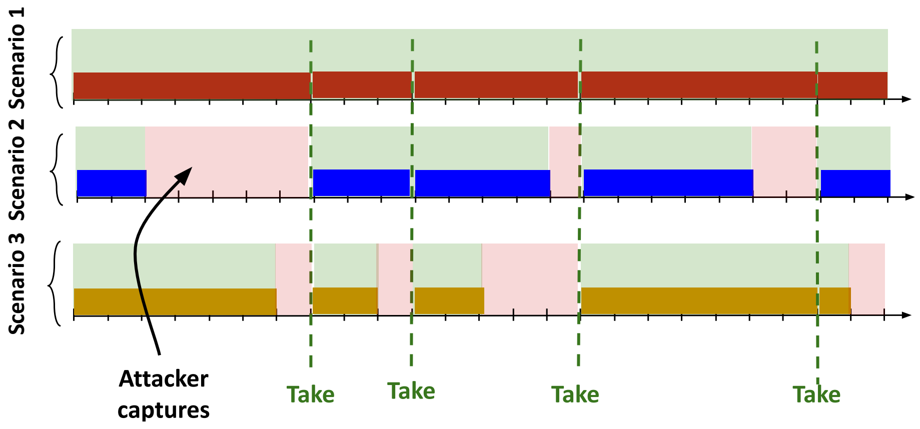


Figure 2.11: A single defensive schedule for multiple attack scenarios.

The model below formulates `PLSCH` as the described job assignment problem. It takes (i) a number of machines with different sequences of jobs (potential attack scenarios in the original problem) and (ii) a fixed defender budget as input. It results in a schedule for the multiple-machine job assignment problem, which represents an MTD schedule of *take* actions. The model uses two decision variables, x_t and y_{mjt}. x_t is a binary decision variable that represents if a *take* action is scheduled at the time instance $t < T$. y_{mjt} is the other binary variable to decide if a job $j \in J_m$ of machine $m \in M$ is scheduled to start at the time instance t.

The (objective) Function 2.23 maximizes the number of assigned jobs to eventually minimize the idle time (minimizing the attacker's control over the system in the PLADD problem). Constraint 2.24 ensures that j can be scheduled only once on m, and Constraint 2.25 ensures that a job k cannot be scheduled on a machine m before all other jobs j in the job set J_m s.t., $j < k$ are placed. Constraint 2.26 ensures that two consecutive jobs j and k s.t. $k = j + 1$ cannot overlap as the successor job k restricted to start after the whole duration of j, d_{mj}, s.t. $t + d_{mj} \leq u$, where t and u are the starting times of j and k, respectively. Those three constraints guarantee that all attacks of a potential attack scenario are considered in the given order, timing characteristics, and interdependencies.

Constraints 2.27 and 2.28 represents the dependencies between two decision variables. Constraint 2.27 ensures that (i) no job j can be scheduled at t unless there is a defender's action s.t., $x_t = 1$, and (ii) at most one job can be placed on machine m at a given time instance t. Complementarily, Constraint 2.28 implies that there should be at least one job scheduled in one of the machines if the defender takes action. Note that these constraints are not directly parts of the original PLADD model, but are related to the representation of the job assignment problem. Lastly, Constraint 2.29 limits the budget β of the defender in terms of the number of actions.

$$\max \sum_{m \in M} \sum_{j \in J_m} \sum_{t=0}^{T} y_{mjt} d_{mj} \tag{2.23}$$

$$\sum_{t=0}^{T} y_{mjt} \leq 1 \qquad \forall m \in M, \forall j \in J_m \tag{2.24}$$

$$\sum_{t=0}^{T} y_{mjt} - \sum_{t=0}^{T} y_{mkt} \geq 0 \qquad \forall m \in M, \forall j, k \in J_m, k = j + 1 \tag{2.25}$$

$$(d_{mj} + t) y_{mjt} + (T - u) y_{mku} \leq T \qquad \forall m \in M, \forall j, k \in J_m, k = j + 1, \forall t, u \leq T \tag{2.26}$$

$$\sum_{j \in J_m} y_{mjt} \leq x_t \qquad \forall m \in M, \forall t \leq T \tag{2.27}$$

$$\sum_{m \in M} \sum_{j \in J_m} y_{mjt} \geq x_t \qquad \forall t \leq T \tag{2.28}$$

$$\sum_{t=0}^{T} x_t \leq \beta \tag{2.29}$$

While this model specifies when to take actions for the defender, i.e., the temporal schedule, it is also required to decide on which service configuration should be set, i.e., the spatial configuration, for *moving* the critical assets within each defensive action. Accordingly, we present the overall spatio-temporal framework, `PLSCH-MTD`, in the next subsection.

PLADD-Scheduling MTD (PLSCH-MTD) Optimization Model

The PLADD-Scheduling MTD (`PLSCH-MTD`) combines `JSAR` (green blocks) and the `PLSCH` (red blocks) as shown in Figure 2.12. It first utilizes `JSAR` to generate a set of feasible service configurations and then `PLSCH` to schedule those configurations within MTD actions. Here, an action is the reconfiguration of services by migrating service instances and rerouting their communication paths. The `PLSCH-MTD` enforces at least a certain amount of changes between successive configurations to increase the chances that the attacker loses its insights on the service configuration. However, excessive changes between two configurations potentially induce reconfiguration overhead and may cause interruption of services. As a result, the *distance between two configurations* deduces a trade-off between defensive capabilities and reconfiguration overhead.

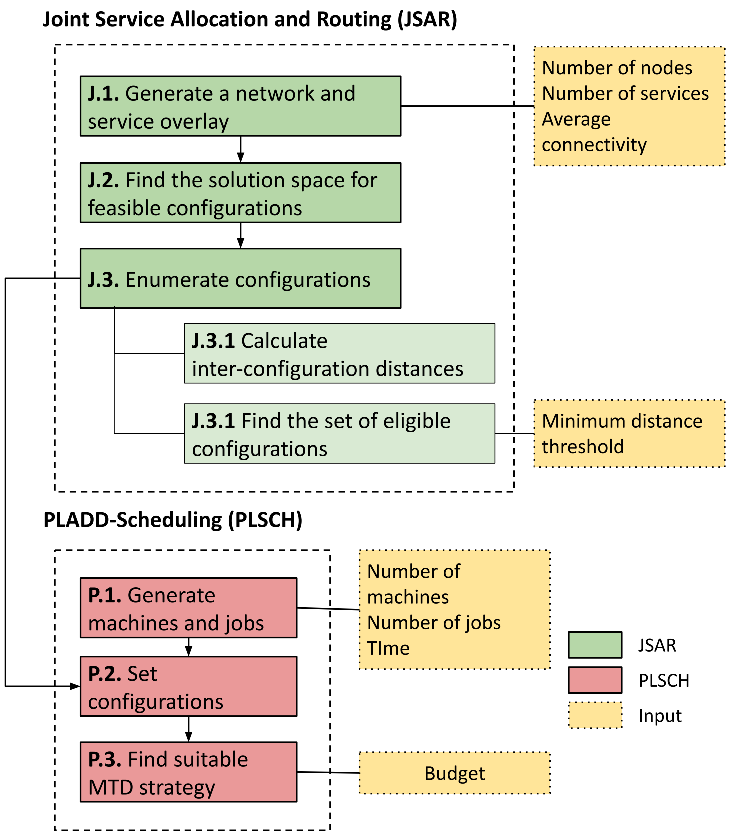

Joint Service Allocation and Routing (JSAR)
J.1. Generate a network and service overlay
J.2. Find the solution space for feasible configurations
J.3. Enumerate configurations
J.3.1 Calculate inter-configuration distances
J.3.1 Find the set of eligible configurations
Number of nodes
Number of services
Average connectivity
Minimum distance threshold
PLADD-Scheduling (PLSCH)
P.1. Generate machines and jobs
P.2. Set configurations
P.3. Find suitable MTD strategy
Number of machines
Number of jobs
TIme
Budget
JSAR
PLSCH
Input

Figure 2.12: The overall optimization framework: `PLSCH-MTD`.

Having a large set of feasible configurations, a defender should decide which configuration it sets next as a defensive action. It requires evaluating which configurations are eligible to be set consecutively so that this reconfiguration imposes a sufficient amount of changes in the system. To measure that, we propose a new metric, *distance*, computed between two configurations c and e as

$$|c - e| = \frac{\sum_{s\in S} \sum_{v\in V} |q^c_{sv} - q^e_{sv}| + \sum_{d\in D} \sum_{p\in P} |z^c_{dp} - z^e_{dp}|}{|S| + |D|} \tag{2.30}$$

In Equation 2.30, q^c_{sv} and z^c_{dp} represent the service deployment and demand assignment variables (see Section 2.1) for the configuration c, respectively. The distance between two configurations c and e is proportional to (i) the number of service migrations, i.e., services migrated to different nodes than the previous configurations, and (ii) reroutings, i.e., traffic streams moved to different paths. Equation 2.30 can also be used to calculate the migration overhead that may cause a certain delay and configuration effort for each reconfigured component.

The PLSCH-MTD calculates the eligibility of each combination of potential configurations $c \in C$ in advance, considering a threshold distance κ given as input. Two configurations can be set consecutively only if there is a sufficient distance in-between s.t., $|c - e| > \kappa$, which is represented as $\alpha_{ce} = 1$. Accordingly, Constraint 2.31 ensures that each action involves two eligible configurations satisfying the given threshold distance as follows.

$$y_{mjt} + y_{mkf} - 1 \leq \sum_{c \in C} \sum_{e \in C} a_{ct} a_{ef} \alpha_{ce}$$
$$\forall m \in M, \forall j,k \in J_m, k = j+1, \forall t,f \leq T \tag{2.31}$$

a_{ct} is a binary decision variable, s.t. $a_{ct} = 1$ if the system has the service configuration c at time instance t as a result of a defender action. Similar to the constraints of the JSAR, this constraint is also linearized to simplify the overall model. Constraint 2.32 ensures that a respective configuration is assigned at t if a defensive action is taken s.t. $x_t = 1$.

$$\sum_{c \in C} a_{ct} \leq x_t \qquad \forall t \leq T \tag{2.32}$$

Lastly, Constraint 2.33 avoids the reuse of the same configuration for the given system duration T to prevent an attacker from estimating the next configuration.

$$\sum_{t=0}^{T} a_{ct} \leq 1 \qquad \forall c \in C \tag{2.33}$$

Attack Scenarios

Several authors of related work tackle single attack scenarios, which cannot provide an optimal MTD schedule against multiple potential attack patterns [TEK14, TMMI16]. More theoretical related work does not reflect realistic attacks well since they only use probability distribution functions for attack generation [JOG+15, JOG+17]. Moreover, data on actual attacks against MCSs is limited to public reports and white papers that partially include attack durations and lack details regarding a complete attack timeline [Pol17]. Therefore, it is difficult to obtain the complete picture of specific attack paths and the duration of advanced attacks. Accordingly, we model different attack types and scenarios considering the recent security incidents in MCSs. An attack scenario is the combination of several individual attack steps. Those scenarios are then used to evaluate the defensive strategies that PLSCH-MTD provides.

Timing characteristics of attack steps: We first define three attack types in terms of their duration: long, medium-length, and short attacks. The length of an attack represents its time-to-success value in the PLSCH model. Moreover, we introduce a new variable, Λ, the *attack scale*, to set the relative lengths of different attacks in proportion to a common design parameter. Accordingly, the length of each attack is uniformly sampled from an interval proportional to Λ. While higher Λ values provide a higher number of shorter attacks, the opposite results in fewer but longer attacks. The attack types are characterized as follows:

- **Long attack:** It represents the longest phases of an attack scenario, e.g., reconnaissance, developing necessary tools, and executing relatively complicated attack steps. The length of long attacks is sampled from the range of $[0.1\Lambda, 0.3\Lambda]$, s.t., it lasts 20% of a scenario with 10% deviation for $\Lambda = T$.
- **Medium-length attack:** It represents a certain number of successive attack steps that require significant time, e.g., encrypting a large amount of data or doing lateral movement across different network components. A medium-length attack is sampled from the interval $[0.05\Lambda, 0.15\Lambda]$.
- **Short attack:** It represents a combination of successive attack steps with short execution time, e.g., changing the configuration of a component, modifying log files, etc. Their length is sampled from the interval $[0.0025\Lambda, 0.075\Lambda]$ taking on average 5% of an attack scenario.

Composition of attack scenarios: We also define four attack scenarios that reflect the characteristics of common security incidents that have recently occurred in critical networked systems. They are composed of the attacks described above in dependence on different attacker goals. Each attack scenario takes at most T time, i.e., contains attack steps in a total duration less than T.

- **Calibrated attacks:** Calibrated attacks target specific components, technologies, and protocols in an MCS. Therefore, they require detailed system-specific knowledge and special exploits that induce long reconnaissance and development times. After acquiring access to the system, the attacker conducts a well-targeted sequence of attacks to potentially multiple components. Accordingly, we compose calibrated attack scenarios of (i) an initial *long* attack and then (ii) randomly selected *medium-length* and *short* attacks.
- **Lateral movement:** After gaining access to the system, an attacker can move laterally through the network to find critical services or sensitive data. While this still requires an initial reconnaissance time, the attacker should also discover further vulnerabilities to continue its lateral movement [WHK+19], which imposes relatively shorter discovery campaigns. Meanwhile, gaining access to the other components potentially requires conducting more spontaneous attacks. Accordingly,

we compose lateral movement scenarios of (i) an initial *long* attack for reconnaissance, (ii) several *short* attacks for exploitation (between one to three attacks in our model), (iii) *medium-length* discovery periods to move laterally, and (iv) repeating (ii) and (iii) steps through the movement.

- **Ransomware:** Ransomware attacks spread a generic malware to encrypt files on the target systems and make them inaccessible. These attacks usually start with a phishing attempt or malvertising [OAMAr21]. Then, the attacker can wait a long time to discover the most sensitive data or cause the most damage to the target system at the right time. Lastly, it requires several operations for encrypting and copying the respective data. Accordingly, we compose these scenarios of (i) a *medium-length* penetration time using one of the mentioned techniques, (ii) a *long*(er) discovery and activation time, and (iii) *short* operations for obtaining encrypted data as many as their total duration stays under T.

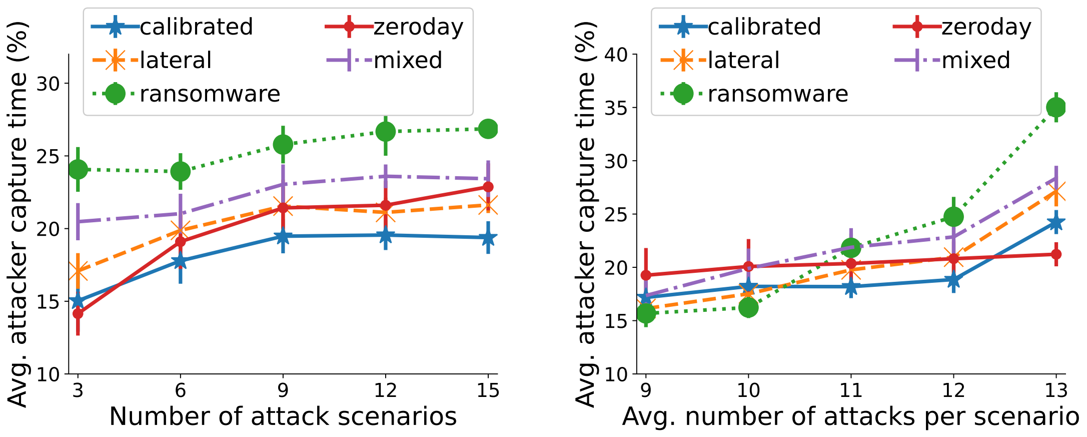
- **Zero-day:** Lastly, zero-day scenarios represent the threats that have not been encountered and thus not analyzed yet. They are composed of randomly-selected *long*, *medium-length*, and *short* attacks with a total duration of T.

Alternatively, it is possible to defend against different scenario types at once, e.g., generating several instances per scenario simultaneously, which is referred to as *mixed* scenarios in the evaluation part. More details and examples of the attack scenarios can be found in the original paper [ESKF23] (Appendix D).

Key Results

We evaluated `PLSCH-MTD` in terms of the attacker's capture time (ACT) for an increasing number of attack scenarios and defender budget. ACT is the percentage of the time that the attacker gains control over the system after a successful attack until the defender takes an MTD action. It is measured by the ratio of the total gap between each consecutive job across all machines in `PLSCH` to the total length of time horizons, i.e., $T * |M|$. We generated many samples of the described attack scenarios to evaluate the challenges that different compositions of varying attacks impose. We also experimented with different values of the inter-configuration distance threshold. As a result, we set it to 15% as it provides several reconfiguration options and still enforces the attacker to rediscover the service configuration. Lastly, we generated random network topologies and service overlays. Our goal in these experiments was to show that `PLSCH-MTD` can provide MTD strategies that protect against several attack scenarios simultaneously, and they can be even more effective proportional to the defensive budget.

Figure 2.13a shows the impact of an increasing number of attack scenarios on the ACT. Regardless of the scenario, we observe only a subtle increase from 1% to 3% in ACT. However, defending against multiple scenarios imposes a base challenge that



(a) Increasing number of scenarios (b) Increasing number of attacks

Figure 2.13: The impact of varying attack scenarios on the ACT.

results in 15-25% ACT. The results indicate that although an MTD strategy remains protective against an increasing number of attack scenarios, it is still difficult to defend against even few concurrent scenarios.

The impact of the type of attack scenarios on ACT is more substantial than the impact of their quantity. In Figure 2.13a, defending against ransomware is the most challenging with 25% ACT since it consists of several short attacks that can be accomplished. Other scenarios are similarly threatening with 15-18% ACT. Therefore, the effectiveness of PLSCH-MTD is highly dependent on the actual attack scenario.

To evaluate the impact of the number of attacks per scenario, we set $T = 60$ and $\Lambda \in [90, 50]$ (in a reversed order). Decreasing Λ shortens the length of individual attacks and increases their number per scenario, which results in 9-13 attacks for the given range of Λ values. Accordingly, Figure 2.13b shows the ACT measurements for increasing attacks per scenario. In the figure, the ACT does not significantly change for 9 to 12 attacks within each attack scenario as there is enough defender budget ($\beta = 12$). This also affirms the results regarding the base challenge (15-25%) of defending against multiple scenarios in Figure 2.13a. However, the attacker's success increases by 5-10% for 13 attacks due to the insufficient defender budget.

Figure 2.14 shows the impact of an increasing defender budget on the ACT for different numbers of attacks per scenario, i.e., for 18 and 12 attacks by setting $\Lambda = \{60, 90\}$. As seen in the figure, more budget strengthens the defender to hold control of the resources with a decreasing ACT regardless of attack counts. When the defender budget is less than the number of attacks per scenario (solid, blue line), we can observe that a gradual increase in the budget decreases the ACT from 35% to 20%.

As shown in Figure 2.14, it is not possible to obtain a complete protection, i.e., 0% ACT, quickly with a linear increase in the budget after the ACT has converged to 10-

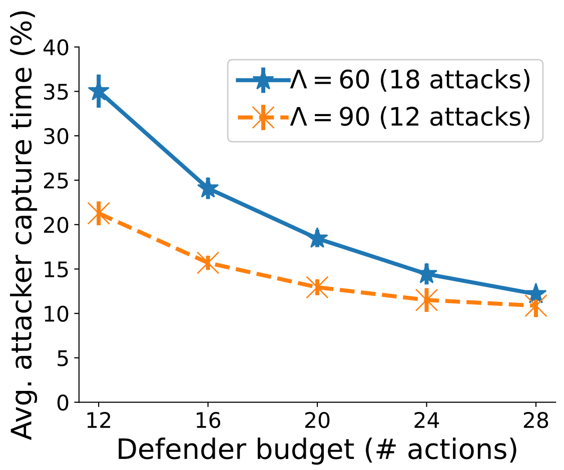


Figure 2.14: The impact of defender budget on the ACT.

12% due to the challenges in defending against multiple attack scenarios. However, PLSCH-MTD can still achieve protection of up to 90% of the system operational time with a defensive budget $\beta \ll T$. Further details on the experimental setup and an in-depth discussion of the results are given in the original paper [ESKF23] (Appendix D).

Takeaways

- Changing the service configuration more often does not simply give a better chance to the defender as proportional to the increasing budget. It may require a significant increase in the budget since, theoretically, any defender budget $\beta \geq T$ should guarantee a complete defender occupation, e.g., when it can change the configuration at every $t \leq T$ with a significant reconfiguration overhead. This answers the `RQ.S3.1`, and also indicates a need for further defensive mechanisms that should be in place together with the MTD strategies.
- It is shown that even a small number of attack scenarios brings a baseline challenge up to 25% ACT. However, rather than their quantity, the type of attack scenarios with a unique combination of different attack types has a more noticeable impact on the attacker's success. Therefore, it is more important to identify an accurate attacker model, e.g., regarding the composition of attack steps in our model, than to be prepared for an increasing number of attack scenarios. This also addresses the `RQ.S3.2` by revealing the effectiveness of the dynamic service configuration against various attack scenarios.

Chapter 3

Resilient Time-sensitive Networking

This chapter presents the main contributions of the thesis on resilient time-sensitive networking for next-generation MCSs. Section 3.1 first briefly introduces the related IEEE 802.1 Time-Sensitive Networking (TSN) protocols that we employ in our contributions. Afterwards, the rest of the sections are organized to gradually develop the configuration and maintenance methods for the resilience of the overall time-sensitive communication. It includes utilizing those protocols *for* resilience of MCSs as well as addressing the potential issues *of* their resilience.

- In Section 3.2, we present a MILP optimization model for joint bandwidth reservation and packet scheduling for the primary time-sensitive traffic shaping protocol, IEEE 802.1Qbv Enhancements to Traffic Scheduling: Time-Aware Shaper (TAS). This model is then deployed in a self-configuration framework that **autonomously** configures time-sensitive streams by excluding the active participation of end-points [SBEF21], and utilized within various re-configuration strategies [SBEF22]. These contributions establish the initial (self-)configuration of networking to fulfill strict timing requirements of critical streams and address the `RQ.T1`.

- In Section 3.3, we propose a graph metric to configure redundant paths reliably for the **fault tolerance** protocol, IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER) [EF21b]. We also develop a framework for the joint use of data and control plane functions to orchestrate redundant paths in FRER [EF21a]. These contributions establish reliability by redundancy over the initial time-sensitive configuration and address the `RQ.T2`.

- In Section 3.4, we first discover the potential security threats against TSN protocols [EBN+21]. Then, we develop an open-source monitoring and intrusion detection system to tackle the identified threats [ESF23]. Those contributions enable **security** monitoring for established configurations and several other security threats and address the `RQ.T3`.

Similar to Chapter 2, we name the sections after their respective resilience goals, i.e., fault tolerance, autonomy, and security. Each section is then structured to present (i) a set of fine-grained research questions, (ii) our related publications, and (iii) the summary of contributions and their takeaways.

3.1 Preliminaries

TSN offers a spectrum of protocols to manage different traffic classes, ensure deterministic communication within a bounded delay, define filtering and networking policies, and improved reliability by using redundant paths. Apart from the core standards, it also includes further protocol extensions, complementary configuration and resource allocation techniques, and network data representation models [FMT+22]. Among more than 30 standards in total (by November 2022), this thesis addresses the following core protocols with impact on the resilience of time-sensitive networks.

1. **IEEE 802.1Qbv Enhancements to Traffic Scheduling: Time-Aware Shaper (TAS)** is the primary protocol for scheduling of mixed-criticality streams. A scheduling protocol is a prerequisite to guarantee deterministic communication and TAS introduces a fine-grained configuration mechanism that helps to minimize the end-to-end jitter.

2. **IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER)** is the only TSN protocol to establish fault-tolerant communication over multiple redundant paths.

3. The **Stream Reservation Protocol (SRP)** is defined and extended in the standards IEEE 802.1Qat and 802.1Qcc. It enables resource reservation and orchestration for the TSN protocols including both TAS and FRER.

A common terminology is used in all three protocols concerning the actors of time-sensitive networks. A *TSN bridge* is fundamentally an Ethernet switch to exchange packets between endpoints and other bridges utilizing TSN protocols. A *talker* is the source endpoint that the time-sensitive traffic is originated from. Similarly, a *listener* is the destination endpoint in a time-sensitive communication. Lastly, a *TSN stream* is a data flow between a talker and a listener and uniquely identified using different identification functions. In the rest of this section, we introduce the relevant mechanisms of TAS, FRER, and SRP protocols are given in more detail. Besides, at the end of the section, a summary of their use in the contributions described in this chapter is given.

IEEE 802.1Qbv Enhancements to Traffic Scheduling: Time-Aware Shaper

TAS schedules frames of mixed-criticality streams per their priority class in fixed-length time windows. This is very similar to traditional time-division multiple access (TDMA) algorithms [IEE16]. TAS models eight different priority queues, i.e.,

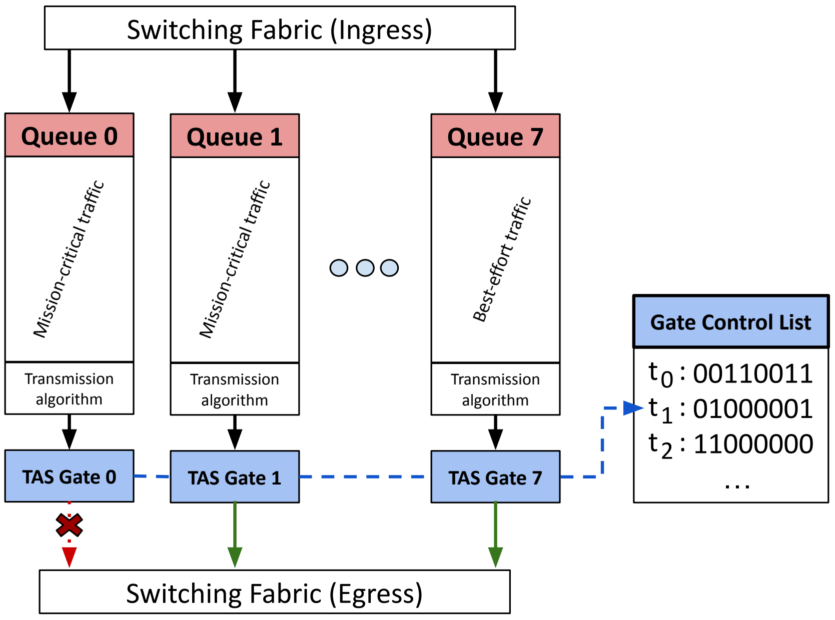


Figure 3.1: The basic structure of a TSN bridge with the TAS protocol.

a queue per TSN traffic class, with transmission gates that are controlled by a Gate Control List (GCL). Periodically, i.e., for each consecutive time window, the GCL opens the selected transmission gates according to the loaded configuration. Figure 3.1 shows the overall structure of this queueing mechanism. While the first two queues are responsible for the transmission of mission-critical traffic, the last one is for best-effort streams, which do not have time-sensitive requirements. The gate control list is configured at the window t_1 to allow only TAS gate 0 and 7 to be open (solid, green lines) and the other is closed (dashed, red line) for the given time instance of the network. Per queue, a transmission selection algorithm decides on which frames should be forwarded first as there may be several streams belonging to the same traffic class.

The transmission rate for a particular traffic class mostly depends on how often the respective gate is open. Therefore, the configuration of the GCL is decisive on the reserved bandwidth and also the time scheduling. Note that the overall configuration of the network still requires that all participants of TSN networks should be synchronized in time to guarantee the end-to-end deterministic communication.

IEEE 802.1CB Frame Replication and Elimination for Reliability

FRER is the primary solution for TSN to tolerate link and node failures. For that, it offers a static redundancy mechanism by replicating Ethernet frames, via multiple, preferably node-disjoint redundant paths. FRER also comes with an elimination

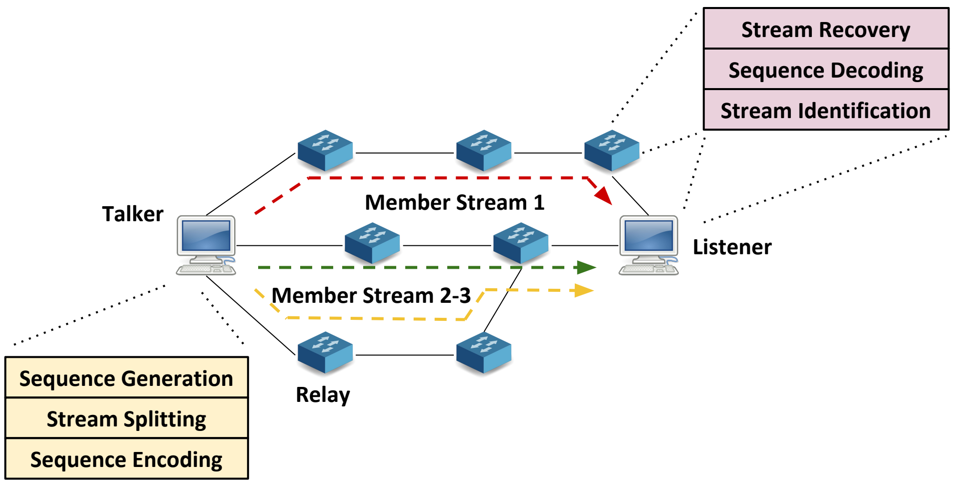


Figure 3.2: An example deployment of FRER.

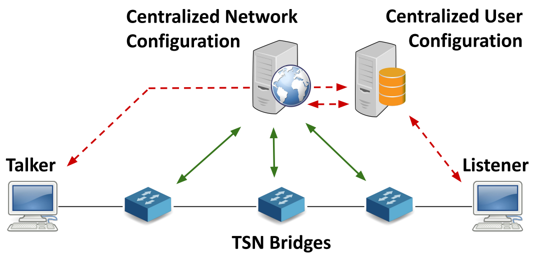
mechanism that runs on TSN bridges and end-hosts to drop replica packets. This mechanism avoids redundant transmissions across the same link and protects the network against loops and babbling idiots, e.g., stuck senders sending the same packets unintentionally [IEE17b].

Figure 3.2 shows the use of FRER in a network where three disjoint paths are assigned to a stream, e.g., two of them for redundancy. In the figure, both listener and the bridge that forwards two member streams can drop redundant packets. Generally, the talker performs:

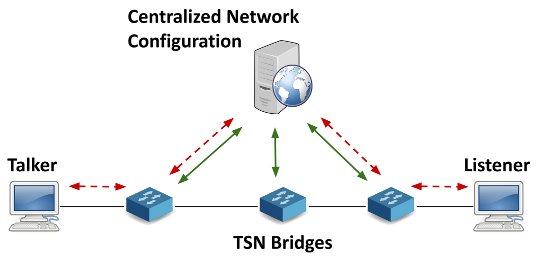
- *sequence generation* by generating a unique identifier per packet of a stream to be incremented for the other packets in the sequence.
- *stream splitting* by copying the packets and creating member streams to be sent through k distinct paths.
- *sequence encoding* by assigning a sequence number to the copied packets via the so-called Redundancy Tag (R-TAG).

A listener or a bridge performs:

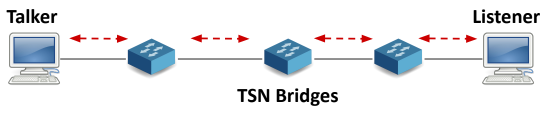
- *stream identification* by applying a stream identification function, e.g., a function taking destination MAC and VLAN ID of a packet as input, to distinguish a stream.
- *sequence decoding* by extracting the sequence identifier of a packet to be compared to the identified stream's sequence information.
- *stream recovery* by deciding if a packet is duplicate and should be dropped or forwarded, and lastly


Centralized Network Configuration
Centralized User Configuration
Talker
Listener
TSN Bridges

(a) Centralized



(b) Hybrid



(c) Distributed

Figure 3.3: Different SRP configuration models.

- *latent error detection* by counting if it has received the expected number of duplicate packets to detect a node or link failure on the path of a member stream.

On the receiver side, the stream recovery stage consists of two functions. The sequence recovery function (SRF) processes all the packets received from different ports of the switch, and thus, it can detect duplicate packets of a stream coming from different paths. In contrast, the individual recovery function (IRF) processes the stream coming from a single path (or port) and is effective against, for instance, duplicate packets due to a stuck sender. Any recovery function utilizes an algorithm to decide on packet forwarding or dropping. A timeout duration is set for that function to reset the expected sequence number (and interval) to refresh the recovery function in case of not forwarding any packet for the specified duration due to occasional failures.

IEEE 802.1Qcc Stream Reservation Protocol

SRP introduces the resource reservation routines for time-sensitive streams to configure all TSN components in the systems satisfying tight QoS requirements. It proposes (i) a fully centralized network configuration (CNC) entity to configure the TSN bridges in a network remotely and (ii) a centralized user configuration (CUC) entity, which is responsible for the discovery of endpoints [IEE18]. Accordingly, it offers different configuration schemes that utilize those entities.

Figure 3.3 illustrates (a) centralized, (b) hybrid, and (c) distributed models. In the fully centralized model, endpoints directly communicate with CUC over a user/network interface (UNI) (red lines) and request network resources for TSN streams with certain requirements such as the worst-case latency and inter-arrival times.

CNC then configures the bridges according to the requests received by CUC (solid, green lines). Figure 3.3b shows the hybrid centralized network/distributed user model. Here, the edge TSN bridges, e.g., bridges that endpoints are directly attached to, forward SRP requests to CNC with network-wide visibility (dashed, red lines). Similar to the fully centralized model, it is then responsible for the configuration of all TSN bridges (green lines). Both centralized models enable the use of all TSN protocols that potentially requires network-wide configuration. Lastly, in the distributed model shown in Figure 3.3c, TSN bridges forward SRP requests of endpoints to each other to make necessary configurations individually. It is thus restricted to configure only locally-optimum settings.

The SRP provides significant configuration flexibility over many parameters of time-sensitive communication. Accordingly, it has a complex packet structure that allows an endpoint to specify various requirements via type-length-value (TLV) fields and recursive header groups. In the respective standard [IEE18] at Section 35.2 (p.105-134), all possible fields and the concrete packet structure are explained in detail.

3.2 Autonomy

IEEE 802.1Qbv TAS is the most prominent TSN protocol to establish deterministic end-to-end communication. This section introduces a self-configuration framework to optimally schedule critical streams via TAS for addressing the `RQ.T1`.

IEEE 802.1Qbv TAS typically requires tedious configuration for both resource allocation and packet queueing, which is not possible to perform manually for large-scale and heterogeneous time-sensitive systems. While some solutions provide joint scheduling and resource reservation for various channel partitioning protocols, TAS requires the configuration of its gate control list, which is relatively more complex. Besides, its configuration should quickly adapt to changing network conditions without degrading service quality. Therefore, the following questions arise in addition to the `RQ.T1`:

- `RQ.T1.1`: How to handle scheduling and routing time-sensitive streams optimally, maximizing the service delivery ratio?
- `RQ.T1.2`: What are the efficient autonomous reconfiguration strategies for time-sensitive streams in changing network conditions?

Accordingly, the following publication presents the Time-sensitive Optimal Routing (`TSOR`) model to reserve sufficient link resources for the given time-sensitive streams jointly and also finds an optimal gate configuration for TAS:

```
N. Sertbaş Bülbül, Doğanalp Ergenç, M. Fischer. SDN-based Self-
Configuration for Time-Sensitive IoT Networks. IEEE International
Conference on Local Computer Networks (LCN), 2021.
```

It further proposes a self-configuration framework that enhances edge TSN bridges to automatically learn stream characteristics, so that `TSOR` can provide optimal TAS configurations according to the stream requirements[1].

Adapting stream scheduling and routing to changing network conditions leads to sub-optimal configurations in terms of QoS. Therefore, the following publication proposes alternative reconfiguration strategies that utilize several variations of `TSOR`[2]:

[1]In the first publication, the contributions of this thesis are modeling and implementing `TSOR` as well as its complexity analysis and integration to the self-configuration framework. The first author designed this framework, implemented a simulation model, and evaluated the overall proposal.

[2]In the second publication, the contribution of this thesis is designing and implementing restricted and unrestricted variations of the `TSOR`. The first author proposed the overall idea, implemented the other two variations in a simulation model, and conducted the evaluation. The third author helped to improve the quality of the papers with his valuable feedback.

N. Sertbaş Bülbül, Doğanalp Ergenç, M. Fischer. Towards SDN-based Dynamic Path Reconfiguration for Time-sensitive Networking. IEEE/IFIP Network Operations and Management Symposium (NOMS), 2022.

In the remaining of this section, Section 3.2.1 presents the details of the TSOR together with its employment in our self-configuration model. Then, Section 3.2.2 shortly introduces different stream reconfiguration strategies. The respective publications [SBEF21] and [SBEF22] are given in Appendix E and F.

3.2.1 SDN-supported Self-configuration of IEEE 802.1Qbv TAS

Our self-configuration framework, SC-TSN, removes the end-host-related dependencies of TSN configuration. As described in Section 3.1, conventionally, end hosts should actively announce their communication requirements to a centralized entity using IEEE 802.1Qcc SRP, so that a controller (or directly TSN bridges) can perform resource allocation and scheduling. Instead, in SC-TSN, TSN bridges on the network edge automatically learn stream characteristics and assist an SDN controller in configuring the rest of the TSN bridges. The SDN controller utilizes TSOR to allocate sufficient resources and to schedule time-sensitive streams via IEEE 802.1Qbv TAS optimally.

Although SC-TSN does not remove the dependency on a controller, it offers a seamless configuration technique and allows low-power talkers and listeners, e.g., sensor modules and robotics, to be deployed without explicit TSN support. It eventually enables the overall system to configure itself autonomously.

The SC-TSN framework has two main mechanisms: stream learning and stream placement. When a talker starts to send a stream, the respective edge TSN bridge derives the date rate and volume of that stream and notifies this information to the SDN controller. Meanwhile, the stream is forwarded through a preconfigured path with only best-effort guarantees. The SDN controller acts as a CNC (in the context of IEEE 802.1Qcc SRP, see Section 3.1) with network-wide visibility. If the stream is deducted as a high-priority and critical flow, the controller runs the TSOR model to allocate resources for the optimal end-to-end path and configure the IEEE gate lists 802.1Qbv TAS on every TSN bridge. Then, the stream is migrated to its dedicated path.

As it is the main configuration of this thesis, the following section focuses on the TSOR model. More details on the SC-TSN framework can be found in the original paper [SBEF21] (Appendix E).

Time-sensitive Optimal Routing (TSOR) Model

The `TSOR` is a MILP model that finds optimal end-to-end paths for given traffic streams. It includes allocating required resources for each link and configuring the TAS gate lists for each TSN bridge on that path. Its overall objective is minimizing the latency for critical streams. Given that the working principles of TAS are given in Section 3.1, the rest of this section presents the formulation of `TSOR`.

$$\min \sum_{d \in D} \sum_{p \in P_d} \sum_{e \in E} \sum_{s \in S} x_{dp} \alpha_{ep} \left[l_e^o + l_e^q (1 - g_{es}) \right] \tag{3.1}$$

$$\sum_{p \in P_d} x_{dp} = 1 \qquad \forall d \in D \tag{3.2}$$

$$\sum_{d \in D} \sum_{p \in P_d} x_{dp} \alpha_{ep} h_d \leq c_e \qquad \forall e \in E \tag{3.3}$$

$$\sum_{s \in S} g_{es} = 1 \qquad \forall e \in E \tag{3.4}$$

$$\sum_{p \in P_d} \sum_{e \in E} x_{dp} \alpha_{ep} \left[l_e^o + l_e^q (1 - g_{es}) \right] \leq l_d \qquad \forall d \in D \tag{3.5}$$

$$g_{es} - \sum_{d \in D} \sum_{p \in P_d} x_{dp} \alpha_{ep} \frac{h_d}{c_e} \geq 0 \qquad \forall e \in E, \forall s \in S \tag{3.6}$$

The `TSOR` has two optimization variables: x_{dp} and g_{es}. x_{dp} is a binary variable to decide if stream $d \in D$ is assigned to directed path $p \in P_d$. Here, each d is defined between a talker and a listener, where D is the set enumerating all streams. Accordingly, P_d represents the set of paths computed between those two particular endpoints. g_{es}, is a continuous variable defined in $[0, 1]$ and represents the opening frequency of a gate on the egress port of link $e \in E$ for the service class s. While $g_{es} = 1$ infers that the gate for s should be open all the time and the capacity of the entire link e is used for the streams of service class s, $g_{es} \approx 0$ means that any stream of service type s is not active at all on the respective port and thus, the gate can stay closed. Eventually, the respective gate for the service class s on link e is open as proportional to $0 < g_{es} < 1$. From this perspective, g_{es} is dependent on the total resources reserved for the streams of service type s.

The (objective) Function 3.1 minimizes the overall latency in the selected paths. The variables in the objective function are explained in the context of the latency Constraint 3.5. Constraint 3.2 ensures that each (non-bifurcated) stream $d \in D$ is assigned to exactly one path $p \in P_d$. Constraint 3.3 guarantees that each link e has sufficient capacity c_e to handle the total load h_d of all streams $d \in D$ assigned to any path p including e, s.t. $\alpha_{ep} = 1$. Constraint 3.4 limits the opening frequencies of gates on egress port (or link) e since only a set of them can be practically open at the same time proportional to stream load per service class s. Constraint 3.5 ensures that the end-to-end latency on path p is always smaller than the tolerable latency for stream d, which is l_d. Besides, the gate configuration g_{es} on the respective egress port of each link e that belongs to path p, s.t. $\alpha_{ep} = 1$, impacts the end-to-end latency. Constraint 3.6 forces g_{es} to be proportional to the total traffic load of service type s forwarded through link e. Note that lower values of g_{es} increase latency at link e, as it enables forwarding the traffic of service type s less often. Accordingly, we weight the closing frequency $1 - g_{es}$ with the delay factor l_e^q to represent the queueing delay. Beyond that, we added a base delay l_e^o representing the port and link characteristics, e.g., packet processing and propagation delay, to each link. Those design parameters, l_e^q and l_e^o, can be set according to the system and network properties.

$$x_{dp} \geq a_{dp} \qquad\qquad \forall d \in D, \forall p \in P_d \tag{3.7}$$

Lastly, Constraint 3.7 is an additional pre-assignment constraint that helps to fix the streams that are already assigned to a particular path p, i.e., $a_{dp} = 1$ from an existing configuration. a_{dp} is given as input to the problem. For instance, using this parameter, the existing configuration of high-criticality streams can be held intact when a new flow is scheduled.

Key Results

We evaluated the overall SC-TSN framework in terms of the mean and maximum end-to-end latency of time-triggered (TT) streams and the ratio of the delayed TT frames, which arrive later than their deadline. It is then compared with the native SRP protocol, where talkers announce their stream requirements, and a centralized controller still uses `TSOR` to configure the TSN bridges. Accordingly, `TSOR` is the primary configuration model for both methods, and the experiments allow to quantify the overhead of the autonomous configuration process. Our primary goal in the experiments was to achieve the QoS in terms of end-to-end latency equal to the built-in TSN configuration method (SRP) without introducing a substantial self-configuration overhead. The self-configuration framework is implemented in the network simulator OMNeT++, and `TSOR` is implemented in the optimization tool CPLEX and then integrated into the simulation model. Further details on the simulation parameters and considered network topologies can be found in the original paper [SBEF21] (Appendix E).

In the experiments, we examined the impact of increasing interarrival times of the best-effort (BE) frames, from 10ms to 1000ms. It represents the rate of background traffic, which may cause congestion and render scheduling harder. We selected half of the streams as critical TT and the other half as BE with more relaxed latency requirements.

Table 3.1: Mean and maximum latency of TT frames in milliseconds for varying BE traffic rate.

	SRP		SC-TSN	
BE rate	**Mean**	**Max.**	**Mean**	**Max.**
10ms	1.31	10.52	1.35	17.30
20ms	1.30	4.31	1.32	11.44
50ms	1.29	2.67	1.30	8.08

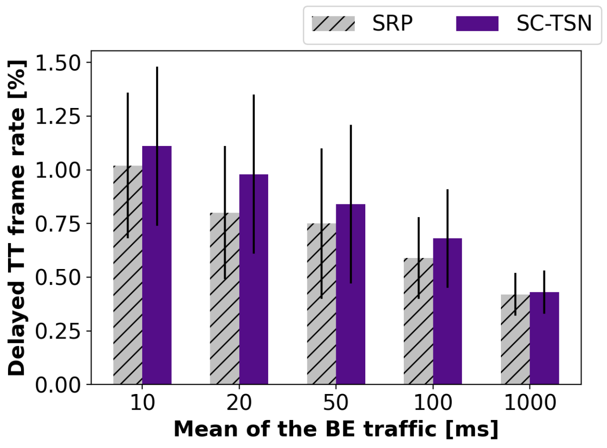


Figure 3.4: The percentage of delayed TT frames.

Table 3.1 shows the mean and maximum latency of TT frames for varying BE load. SRP and SC-TSN have nearly the same mean latency as both utilize `TSOR`, which provides the optimal scheduling. However, SC-TSN induces a higher maximum latency than SRP since all streams are forwarded through a default and potentially sub-optimal route until the respective edge bridges extract their characteristics.

Figure 3.4 shows the percentage of the delayed TT frames with a 95% confidence interval and also confirms the results in Table 3.1. For all varying BE loads, SC-TSN results in a slightly higher, i.e., up to 0.25%, delayed frame ratio than SRP. That small number of latent frames are the ones forwarded through the default path at the beginning. Further results regarding the impact of the learning framework on BE traffic can also be found in the original paper [SBEF21] (Appendix E).

Takeaways

- The enhancement in TSN bridges for automatically extracting stream characteristics enables further autonomy in configuring time-sensitive streams. Besides, it introduces only a slight delay in the delivery of the first few frames of a stream. Therefore, its overhead is negligible, especially for long-lasting streams. Meanwhile, our model, `TSOR`, can handle the resource reservation and scheduling of the streams optimally using these extracted (or learned) characteristics.

- A self-configuration method like SC-TSN should be employed as a part of a hybrid configuration scheme, where the most critical streams that cannot tolerate additional jitter are manually configured or registered via SRP. The rest of the arriving streams can still be placed autonomously. Those two takeaways reveal the offerings and the limitations of a self-configuration scheme and thus answer the `RQ.T1.1`.

3.2.2 Dynamic Reconfiguration Strategies for IEEE 802.1Qbv TAS

The `TSOR` model enables assigning upcoming time-sensitive streams continuously to optimal paths *among the available ones* when utilized by the SC-TSN framework. Accordingly, it keeps the configuration of the active streams stable, i.e., TAS schedules in bridges and link resource reservations, which prevents potential interruptions in critical streams. However, it also hinders the flexibility in the overall configuration that potentially remains sub-optimal in terms of QoS and resource utilization. This section proposes four variations of `TSOR` as different reconfiguration strategies. They differ in terms of reconfiguration frequencies, triggers, and amount of reconfigured streams. Table 3.2 presents a summary of all strategies regarding their flexibility in resource use, resulting QoS, and reconfiguration overhead, which are explained below in more detail.

Table 3.2: The summary of reconfiguration strategies.

Strategy	**Reconfiguration Approach**	**Flexibility**	**QoS**	**Overhead**
`TSOR-R`	Additive configuration	Low	Low	Low
`TSOR-U`	After every stream	High	High	High
`TSOR-P`	After every *k*th stream	Medium	Medium	Medium
`TSOR-T`	Exceeding threshold	Medium	Medium	Medium

1. `TSOR-R`: This strategy fixes the configuration of all active streams and finds the optimal scheduling for an upcoming stream within the remaining network resources using `TSOR`. As it restricts the (re)configuration of active streams, the strategy is named as restricted `TSOR`: `TSOR-R`. It eventually results in a stable configuration with a minimum reconfiguration overhead. However, `TSOR-R` can utilize only leftover resources from the existing streams to *add* a new one, which leads to sub-optimal QoS and resource utilization.

2. `TSOR-U`: This strategy reconfigures all streams together without any restriction using `TSOR` whenever a new stream is registered. Therefore, it is named unrestricted `TSOR`: `TSOR-U`. It eventually finds the optimal configuration for all streams, but potentially requires reconfiguring several active streams, which imposes a trade-off between QoS and reconfiguration overhead.

3. TSOR-P: This strategy reconfigures all streams after the registration of every *k*th stream, which leads to fewer reconfigurations but still converges to the optimal QoS. Since it imposes a (potentially irregular) reconfiguration period depending on the arrival rate of new streams, it is named as TSOR-P. It eventually offers better resource utilization and flexibility than TSOR-R and less reconfiguration overhead than TSOR-U depending on the value of *k*.

4. TSOR-T: This strategy computes both TSOR-U and TSOR-R, and applies the solution of TSOR-U only if the difference between the objective values of those two strategies exceeds a predefined threshold value. It is named as TSOR-T, referring to the threshold value. It eventually evaluates if the potential improvement in QoS in terms of the overall latency is substantial after a network-wide reconfiguration.

Note that those strategies can handle further changes in stream configuration such as removing and modifying the existing ones (beyond placing upcoming streams as described above). For instance, TSOR-U can reconfigure the system after every stream removal. Similarly, TSOR-P can be adapted to run every *k*th stream event including both registration and unregistration.

Key Results

We evaluated our reconfiguration strategies in terms of their stream acceptance ratio and reconfiguration overhead. The acceptance ratio is the proportion of the configured streams, i.e., with reserved resources and scheduling, to all registered streams. It implicitly represents the efficient use of available resources, as additional streams can be placed on suitable end-to-end paths only if there are sufficient resources. For the evaluation, we used the same simulation setup in Section 3.2.1, whose details can be found in the original paper [SBEF22] (Appendix F). In the simulations, we gradually added time-triggered (TT) streams to the network alongside a base (best-effort) data traffic to demonstrate a dynamically increasing communication demand in a system with limited resources. Our main goal in the experiments was to reveal the tradeoff between the acceptance ratio (better use of available resources) and the reconfiguration overhead.

We first compared the acceptance ratio for TSOR-R and TSOR-U, as they represent the least and the most flexible strategies in terms of resource usage. Our evaluation reveals that although it decreases for both strategies with an increasing number of TT streams, TSOR-U achieves up to 5% more acceptance ratio depending on the stream requirements in terms of increasing data rate and delay tolerance.

The difference in the reconfiguration overhead is more significant than the acceptance ratio. Figure 3.5 (top) shows the reconfiguration overhead of all strategies for the same number of TT streams. In the figure, the reconfiguration ratio is the

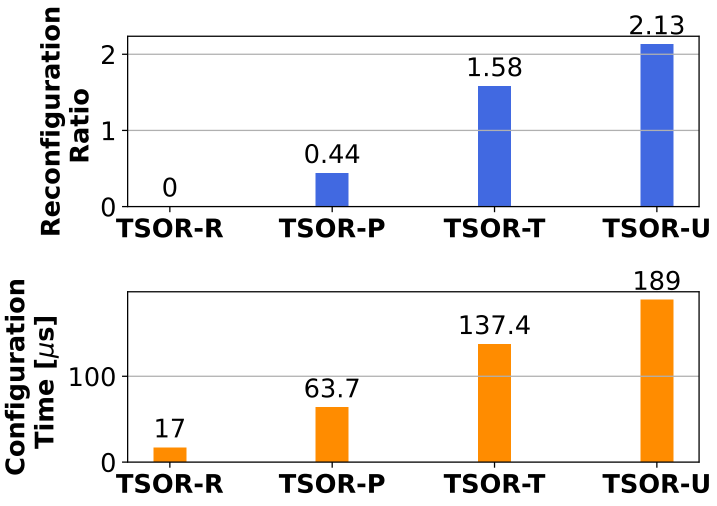


Figure 3.5: Reconfiguration overhead regarding reconfiguration ratio and configuration time.

proportion of the reconfigured streams to the number of total streams. The configuration time represents the time elapsed until the communication starts after a reconfiguration. While `TSOR-R` has no reconfiguration ratio as it does not reconfigure the active streams (other than the upcoming one), `TSOR-U` has the highest ratio since it potentially redistributes several streams after each new arrival. In contrast, `TSOR-T` causes fewer reconfigurations since it performs only if there is a significant QoS improvement. Note that the performance of `TSOR-P` and `TSOR-T` can vary depending on the selection of their design parameters. Decreasing threshold and k values in those strategies would approximate the results to `TSOR-U`.

The configuration time is directly affected by the number of reconfigurations, i.e., including all stream migrations to place an upcoming one. Therefore, the results for the configuration time in Figure 3.5 (bottom) show a similar pattern with the reconfiguration ratio. Frequent reconfigurations in `TSOR-U` result in a high configuration time, while fewer reconfigurations in `TSOR-R` cause less overhead. When interpreted together with the acceptance ratio measurements, these results show that while more flexibility in reconfiguration, i.e., the unrestricted configuration in `TSOR-U`, results in a slightly better acceptance ratio, it also induces a significantly more overhead compared to a restricted one, i.e., `TSOR-R`. As expected, `TSOR-P` and `TSOR-T` perform in between the restricted and unrestricted configurations. More detailed scalability and QoS analysis in different scenarios can also be found in the original paper [SBEF22] (Appendix F).

Takeaways

- Since the `TSOR` does not directly optimize resource allocation but end-to-end latency, the difference between acceptance ratios between different reconfiguration strategies is not substantially high, even for tighter resources. This difference mainly stems from a broader search space for scheduling in unrestricted `TSOR`. It would be possible to deploy even more streams, i.e., higher acceptance ratio, with potentially better resource efficiency and worse QoS when the objective function of `TSOR` is adapted accordingly. In this sense, such changes in the formulation of `TSOR` can also help to develop further reconfiguration strategies.

- Achieving up to 5% better acceptance ratio costs nearly ten times higher configuration time in our evaluation scenarios. Accordingly, the selection of a reconfiguration strategy requires considering this trade-off, which might be relevant to the scalability (placing more streams with a better acceptance ratio) and availability (less interruption time due to reconfiguration overhead) requirements of the target system. Besides, these results depend on several aspects like available resources, QoS requirements of time-sensitive streams, and the amount of background traffic. The design parameters of the selected strategies should also be fine-tuned according to the forementioned requirements and network aspects. These takeaways also address the `RQ.T1.2`.

3.3 Fault Tolerance

IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER) is the primary fault tolerance solution among the current TSN protocols. It offers seamless redundancy to cope with link failures by replicating time-sensitive Ethernet frames on multiple redundant paths. This section introduces novel methods to select the most reliable redundant paths and the best practices to configure FRER, and answers the `RQ.T2`.

In contrast to the existing redundancy mechanisms such as Parallel Redundancy Protocol (PRP) and the Highly-available Seamless Redundancy (HSR) [Net16], FRER does not impose an explicit topological constraint on MCSs, e.g., having isolated network partitions or a ring topology. However, poorly selected redundant paths can inhibit fault tolerance and efficiency. For instance, when the configured redundant paths intersect, FRER's elimination function on the junction bridge could unintentionally drop the duplicate frames. It results in the degradation of the degree of redundancy and renders time-sensitive communication vulnerable to link failures.

While the complexity, overhead, and performance of FRER have been studied before, the redundant path selection and configuration issues have been overlooked. Therefore, the following questions arise in addition to the `RQ.T2`:

- `RQ.T2.1`: How to select suitable redundant paths for the reliable configuration of FRER?
- `RQ.T2.2`: What are the best practices for the configuration and orchestration of FRER?

Accordingly, the following publication first identifies an implicit topological constraint of FRER regarding the selection of redundant paths. It then proposes a path selection metric, `reassurance`, and also an improvement on FRER's frame elimination mechanism to prevent unintentional drops of redundant frames and to maximize fault tolerance against random link failures[3]:

> Doğanalp Ergenç, M. Fischer. On the Reliability of IEEE 802.1CB FRER. IEEE International Conference on Computer Communications (INFOCOM), 2021.

Although FRER provides all necessary functions for redundant communication (see Section 3.1), it does not impose any *internal* configuration for those functions on a TSN bridge. They could be enabled or disabled, or can be performed in different

[3]In both publications, the whole contribution belongs to this thesis. The co-author helped to improve the quality of the papers with his valuable feedback.

orders. Besides, discovering alternative redundant paths and their orchestration still require an *external* network discovery and management mechanism. However, similar to its internal configuration, such mechanisms are not specified in the FRER standard either. Accordingly, the following publication presents the implementation details of the internal and external configuration of FRER, and presents an open-source simulation framework[3]:

```
Doğanalp Ergenç, M. Fischer.  Implementation and Orchestration of
IEEE 802.1CB FRER in OMNeT++.  IEEE International Conference on
Communications (ICC), Workshop on Time-sensitive and Deterministic
Networking, 2021.
```

In the following, Section 3.3.1 introduces the `reassurance` metric and presents the details of an improvement on the elimination function of FRER. Then, Section 3.3.2 shortly introduces a framework for the configuration and management of FRER. The respective publications [EF21b] and [EF21a] are also attached in Appendix G and H.

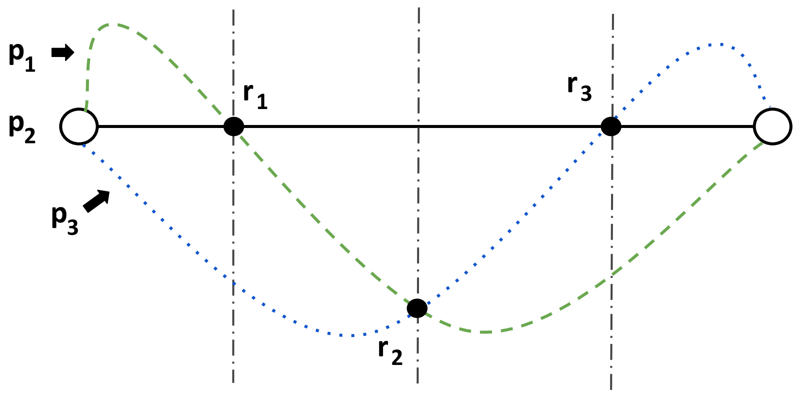
3.3.1 Reliable Path Finding for IEEE 802.1CB FRER

FRER is topology-agnostic and provides significant flexibility for configuring redundant paths. In an ideal scenario, a stream is replicated on k disjoint paths, so that no relay node receives more than one copy of the same frame. For this scenario, a network with a sufficient number of disjoint paths is a prerequisite, which is especially costly for a large number of time-sensitive streams. If the network lacks such paths, we identified the problem that the Sequence Recovery Function (SRF) of FRER may induce unexpected packet drops due to the packet elimination on *junction nodes*, which lies on the intersection of multiple paths. Figure 3.6 shows an edge-case scenario where a talker and a listener communicate via three partially overlapping paths, p_1, p_2, and p_3 (black, green dashed, and blue dotted lines, respectively). r_1, r_2, and r_3 represent the junction nodes, i.e, relay bridges where multiple paths intersect.

In the figure, there are four stages ((a) to (d)) with respect to the junction nodes. The number of frames in each redundant stream is investigated at the different stages. While the numbers at the top show the distribution of traffic in each path by stage, the ones at the bottom show the number of packets in the original stream protected against one or two random failures as well as the number of the original frames that have replicas on redundant paths. Note that the latter is expected to be 100, as all packets should be protected by redundancy.

Stage (a): Initially, the stream has 100 frames replicated to p_1, p_2, and p_3, i.e., the degree of redundancy is three, in stage (a). All packets are replicated three times on this setting, which can tolerate one or two link failures that may occur in p_1, p_2, and p_3.

	(a)	(b)	(c)	(d)
p_1	100	60	60	60
p_2	100	40	40	0
p_3	100	100	40	40



1-failure	100	100	40	40
2-failures	100	40	40	0
Replicated	100	60	40	0

Figure 3.6: An edge-case scenario with three intersecting paths.

Stage (b): At r_1, only the first arriving packets from p_1 and p_2 will be forwarded and all replicated packets will be dropped. Note that p_1 and p_2 can carry some other streams whose amount and type, e.g., priority class, can affect which path can deliver the packets faster to r_1. Here, we assume that the total load is dynamically changing on both paths and the stream is divided into p_1 and p_2 as 60 (i) and 40 (ii) packets. Those 60 (i) packets in p_1 and 40 (ii) packets in p_2 are different, i.e., have different sequence numbers, since r_1 does not forward the same packet to the same path due to the elimination of replicas. As a result, in stage (b), in case of a failure in p_3, the main stream would be delivered as two distinct member streams via p_1 and p_3, instead of two identical streams via two disjoint paths. In case of a second failure, e.g., at p_1 and p_3, only 40 (ii) packets on p_2 would be delivered. Moreover, at maximum 60 out of 100 (i) packets are replicated on each path.

Stage (c): When the packets are forwarded faster on path p_1, r_2 drops 60 (ii) packets coming from p_3 since it has already received them from p_1 before. Note that the 40 (ii) packets sent via p_2 and p_3 are the same packets as they remained as a result of the elimination due to the traffic in p_1. In case of the worst-case single node/link failure after r_2, i.e., p_1 fails or any two failures, only 40 (ii) packets can be delivered to the destination. Besides, the replicas of only 40 (ii) packets are forwarded through p_2 and p_3 whereas 60 (i) packets in p_1 are unique. Thus, in stage (c), the redundancy drops to 40 (ii) packets in contrast to the initial 100 packets.

Stage (d): As p_2 and p_3 carry the same 40 (ii) packets, r_3 eliminates the packets coming from one of those paths, say p_2. In stage (d), none of the three paths forwards all 100 packets. The overall scheme cannot tolerate two simultaneous failures as the number of replica packets is virtually 0 after the ones on p_2 are eliminated.

This scenario shows that each junction node decreases the expected level of redundancy by eliminating replicas. Therefore, for effective use of FRER in the absence of node-disjoint paths, three important points should be considered:

- **Number of redundant paths:** In the absence of disjoint paths, using more than k paths can still tolerate $k-1$ failures by compensating for potential packet drops at the junction nodes. Therefore, it is important to evaluate the degree of redundancy to obtain the desired level of fault tolerance.
- **Number of junction nodes:** When two paths intersect, the unintended elimination of member streams occurs as shown in the scenario. Therefore, minimizing the number of junction nodes between the selected paths is crucial.
- **Position of junction nodes:** An early intersection of two paths, e.g., within the first few hops, can affect redundancy more severely as replicated packets are eliminated quickly and the stream becomes vulnerable to link failures on the remaining segments of redundant paths. Therefore, the position of junction nodes impacts the number of protected links.

The next section introduces a novel graph metric that considers all three points to evaluate the suitability of selected paths for the expected degree of redundancy.

Reassurance: A Path Selection Metric

The metric `reassurance` allows to select k different paths between two endpoints in a way that their overlap is minimal and a potential junction node is close to the receiver. As a result, FRER-induced packet eliminations can be significantly decreased. The metric is defined as $\tau \in [0.0, 1.0]$ for a set of paths to quantify how close their overlapping part, i.e., junction nodes, is to the respective destination nodes if they are not disjoint. When a junction node is closer to the packet destination, packet eliminations can affect only shorter segments of the paths after the junction node. Thus, FRER can effectively protect the longer segments, and it increases the fault tolerance against random link failures. It is then extended to evaluate all sets of k-paths between two endpoints to find their best k-combination.

Let $G = (V, E)$ be a directed graph with nodes $u, v \in V$ and $e = (u, v) \in E$ is a directed edge from u to v s.t., $e : u \rightarrow v$. A *path* p is defined as a sequence of distinct nodes connected by edges in E s.t., $p = (v_1, v_2, ..., v_n) : \exists e \in E, e : v_i \rightarrow v_{i+1})$. Accordingly, the P_k is the set of all k-combinations of the paths in the graph.

As any junction node can affect the forwarded traffic due to a potential packet elimination, selecting paths with the junction node(s) farthest from the origin of the path minimizes the number of links where a stream can be affected. Calculating the *longest disjoint segment* of a path, $\ell(p, C)$, helps to quantify the probability of unintentional frame elimination.

Definition 1: The longest disjoint segment of path p, $\ell(p, C)$, is defined as the segment of p before the first junction node between p and any other path in a particular k-combination of paths $C \in P_k$. Defining the overall set of junction nodes shared by p and any $p_i \in C$ as $V_p^C = \bigcup_{p_i \in C} p \cap p_i$, the index of the first junction node on p, $t(p, C)$ is:

$$t(p, C) = \min\{i : \forall v_i \in V_p^C\} \tag{3.8}$$

and accordingly, $\ell(p; C)$ is

$$\ell(p, C) = \{v_i : v_i \in p \land i \leq t(p, C)\} \tag{3.9}$$

Having the longest disjoint segments of k redundant paths, the `reassurance` metric helps to calculate the ratio of these segments to the overall length of k paths. This also provides the proportional segments of these paths that would not be affected by unintentional frame eliminations.

Definition 2: Reassurance of a set of k paths $C \in P_k$, $\tau(C)$, is the ratio of the total length of the longest disjoint segments of all $p \in C$ to the total number of distinct edges on those paths. When $|\ell(p, C)|$ and $|p|$ are the lengths of the longest disjoint segment of p and the whole p, respectively, $\tau(C)$ becomes

$$\tau(C) = \frac{\sum_{p \in C} |\ell(p, C)|}{\sum_{p \in C} |p|} \tag{3.10}$$

Furthermore, to find the best set of paths between any two endpoints, i.e., the set with the highest reassurance, all possible combinations of k paths between two nodes u and v should be considered as the following.

Definition 3. Reassurance *between two nodes u* and v, $\tau(u, v, k)$ is defined as the maximum reassurance among all k-combination of paths between u and v, $P_k^{uv} \in P_k$ s.t., $P_k^{uv} = \{p_i = (v_1, v_2, ..., v_j) : (v_1 = u \land v_j = v) \land i \leq k\}$.

$$\tau(u, v, k) = \max\{\tau(C) : C \in P_k^{uv}\} \tag{3.11}$$

Figure 3.7 shows a sample graph with three paths, p_1, p_2 and p_3 between two nodes. r_1 and r_2 are the junction nodes on p_2 and p_3, and p_1 and p_2, respectively. Here,

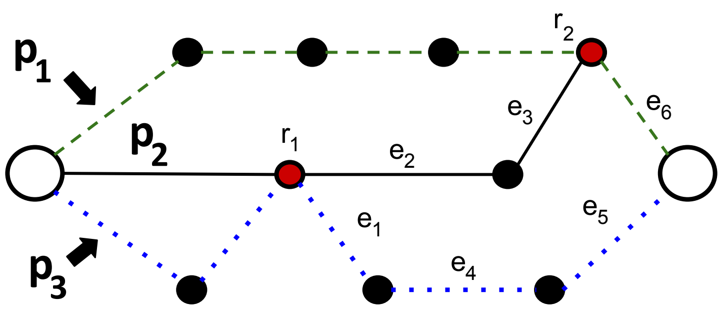


Figure 3.7: Calculation of reassurance for three paths between two endpoints.

any two link failures occured after r_1 and r_2 (namely on links e_1-e_6) can disrupt the communication. For instance, when e_3 and e_6 fail at the same time and r_1 eliminates the traffic coming from p_3, no packets are received by the listener. The length of the longest disjoint segments of p_1, p_2 and p_3 are 4, 1, and 2 (i.e., until r_2 and r_1), respectively. Therefore, the reassurance is $\tau(C) = (4+1+2)/(13) = 0.54$ for $C = \{p_1, p_2, p_3\}$.

Finally, $\tau(u, v, k)$ provides the optimal set of k-paths that minimizes unintentional packet eliminations by considering the required number of redundant paths, the potential number of junction nodes, and positions of those junction nodes. It can be then used as a path-selection strategy to obtain any degree of redundancy in time-sensitive communication.

Enhanced Stream Recovery Function

The main cause of unintended eliminations is that relay bridges are not aware of whether they are junction nodes and drop redundant packets. When a relay can infer its position as a junction node, it can forward a specified number of replicas instead of eliminating them immediately. To establish that, we propose the enhanced stream recovery function (eSRF) (also see Section 3.1 for the original SRF).

First, a bridge should know if it is a junction node for the configured paths. Then, the SRF should forward the first k packets, where k is the junction degree, rather than eliminating them. The junction degree of a bridge represents the number of paths that include this bridge and are assigned to a particular stream. For a stream, if the junction degree of a bridge is 0, it means that the bridge is not on any path used by that stream. If it is 2, for instance, the bridge is expected to receive frames for that stream from two different paths. Consequently, a junction node can evaluate precisely how many replicas it should expect per packet of a stream using the junction degree. Moreover, it can still eliminate the excessive number of replica frames in case of, for instance, maliciously duplicated or misrouted packets after forwarding the expected replicas.

Accordingly, for the reliable use of FRER, we enhance TSN bridges with two modifications.

1. **Detection of junction degree:** In the first modification, a link-state routing protocol, Intermediate System to Intermediate System (IS-IS), is employed in TSN bridges. IS-IS enables link layer routing [FAAS+12] and is the built-in routing protocol for IEEE 802.1aq Shortest Path Bridging (SPB), which is envisioned to be used for network discovery in TSN networks. It enables each bridge to have a network-wide view so that they can compute a set of redundant paths for a stream according to the given metric, e.g., the optimal path combination in terms of `reassurance`. As a result, when a bridge knows the talker and the listener for a stream, and the desired degree of redundancy k, it can automatically compute the potential end-to-end paths and its own junction degree (in case it is a junction node). These information can be sent via SRP (see Section 3.1).
2. **Count of received duplicates:** The second modification includes the deployment of the latent error detection (LED) function of FRER (see Section 3.1) not only on the listener but also on any intermediate relay. The LED normally counts the number of received duplicates received by the listener to make sure that all duplicate packets arrive and there is no failure in the network. Our `eSRF` uses this counter at every bridge to ensure that a relay receives duplicate frames no more than its junction degree. If it receives more, extra frames are still dropped.

As a result, the deployment of IS-IS configured with the `reassurance` metric and the deployment of LED together with `eSRF` improve TSN bridges to detect junction nodes and stop eliminating legitimate duplicate packets.

Key Results

We evaluated the reassurance-based path selection strategy and `eSRF` in terms of the packet delivery ratio for an increasing degree of redundancy (DoR). DoR is the number of redundant paths that are assigned to a stream. We also compared the reassurance-based strategy with alternative strategies: random selection and maximum-disjoint selection, i.e., selecting the paths with the minimum number of intersections. In the evaluation, we injected $k-1$ random link failures for a k-degree of redundancy since it is the maximum number of random failures that the given DoR can tolerate. Our main goal was to show that the reassurance-based strategy can achieve the minimum amount of unintentional packet eliminations and results in the highest packet delivery ratio in case of random link failures in the networks lacking disjoint paths.

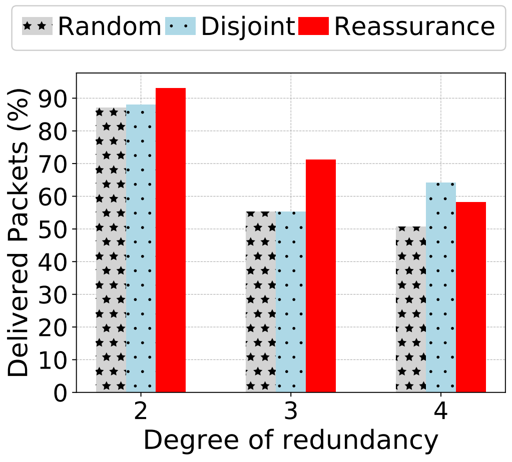
For the experiments, we used a real tier 1 network topology, UUNET[4]. It contains

[4]UUNET and other topology data sets are taken from http://www.topology-zoo.org/dataset.html in the original paper.

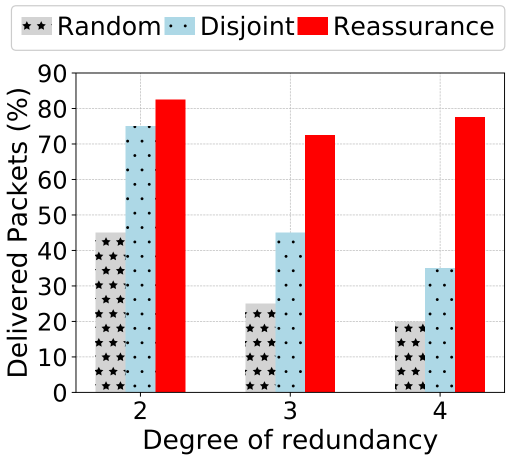
49 nodes with an average node degree of 3,42. Furthermore, to test the performance reassurance-based selection in a network supporting a higher degree of redundancy than UUNET, we also generated smaller but denser random topologies with 30 to 50 nodes and a (fixed) average node degree of 4,2. Note that each node is considered as a TSN bridge, and accordingly two endpoints are connected per bridge to simulate time-sensitive communication between randomly selected endpoints.

Figure 3.8 shows the packet delivery ratio for each path selection strategy in dependence on the DoR, i.e., the number of redundant paths used, for (a) UUNET and (b) random topologies. Figure 3.8a shows the results for $k-1$ random link failures and k redundancy. For $k=2$ and $k=3$, the reassurance-based selection results in the highest packet delivery ratio. Here, the expected number of frames to be delivered is the total number of distinct frames since the replicas are not expected to be delivered in case of $k-1$ failures. For $k=4$, any combination of paths results in a large number of frame eliminations as UUNET's average node degree is less than four and thus the paths are resulting to have a lot of overlap and junction nodes.

Note that no strategy achieves 100% packet delivery *on average*. In the scenarios that a random failure affects (i) multiple paths simultaneously or (ii) the remaining active redundant paths after unintentional eliminations, it is not possible to forward the packets to the destination at all. This occurs more often for increasing number of injected failures (two for $DoR=3$ and three for $DoR=4$). Therefore, in the absence of disjoint paths in the network, packet loss is inevitable when multiple failures occur, and reassurance-based strategy can only offer the best combination among intersecting paths for FRER.



(a) UUNET



(b) Random networks

Figure 3.8: Delivery ratio for increasing DoR. Note that the number of injected link failures also increases with DoR, i.e., $DoR-1$ failures.

Figure 3.8b depicts the results of the same evaluation in random topologies. The results show that with an increasing number of alternative paths, the reassurance-based path selection results in a much higher delivery ratio than random and maximum-disjoint selection strategies. Even for $k = 4$, as the denser topology increases the number of disjoint paths, reassurance achieves a delivery ratio of more than 70%.

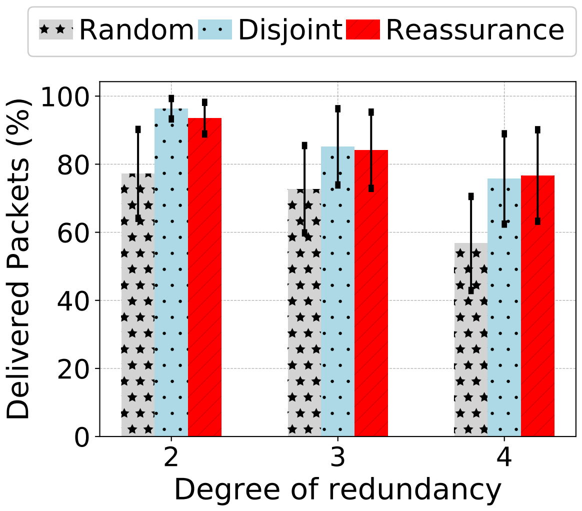


Figure 3.9: Delivery ratio of enhanced sequence recovery function eSRF.

Lastly, Figure 3.9 shows the delivery ratio after we enhanced the relay bridges with Enhanced Sequence Recovery Function (eSRF) in UUNET scenarios. Under $k-1$ failures, maximum-disjoint and reassurance-based strategies offer up to 80-90% delivery ratio. Note that although their packet delivery performances are slightly different, their mostly overlapping confidence intervals show that the eSRF enhancement works equally well for both selection strategies. Besides, while all strategies are still affected by the failures, they perform better in comparison to the results in Figure 3.8a. Note that disjoint paths are required to obtain 100% delivery ratio since no strategy could be effective when a single failure affects multiple paths in those evaluation scenarios. Further analysis of the `reassurance` metric, the implementation details of eSRF, and a more extensive evaluation can be found in the original paper [EF21b] (Appendix G).

Takeaways

- Deployment of fully disjoint paths is obvious and the most reliable solution for redundant communication using FRER. However, in absence of them, it is not enough only to minimize the number of intersections between paths but the position of such junction points should be also considered. Therefore, the redundant paths of FRER should be carefully selected using a comprehensive metric like `reassurance`. This takeaway also answers the research question `RQ.T2.1`.
- Increasing number of alternative paths gives a larger search space to the reassurance-based strategy and results in up to 40% more delivery ratio compared to the re-

sults in sparser networks with less connectivity (comparing Figure 3.8a and 3.8b). Moreover, increasing number of redundant paths even decrease the delivery ratio for other strategies since they do not consider the positions of junction nodes. Therefore, adding more links can only increase fault tolerance when the redundant paths are configured properly.

3.3.2 Configuration and Orchestration of IEEE 802.1CB FRER

IEEE 802.1CB FRER consists of data and control plane functions. While its data plane functions perform stream identification, frame replication and elimination, the control plane functions includes path discovery and orchestration mechanisms to configure redundancy. This decoupled architecture requires configuring FRER (i) internally regarding its data plane functions for frame processing and (ii) externally regarding path finding and assignment for end-to-end communication.

The internal configuration refers to the placement of FRER functions (see Section 3.1) in a logical order in the data plane. While some of those functions such as stream identification and recovery are mandatory, others like latent error detection are optional. Besides, mandatory functions have alternative configurations with different parameters that are adaptable to different network settings. The external configuration includes the integration of routing and path configuration protocols into FRER for the discovery and assignment of redundant paths.

Figure 3.10 shows the relationships between data and control plane functions of FRER over a TSN bridge architecture. We proposed this proof-of-concept architecture and implemented FRER and suitable control plane protocols in a network simulator, OMNeT++ [VH08]. The rest of this section briefly presents the implementation and integration of data and control plane functions, which are essential for the effective use of FRER.

Data plane

The configuration of the data plane consists of stream encoding/decoding, sequence recovery, and latent error detection functions, which take over different roles for the transmission and reception of TSN frames.

Outgoing frames are first sent to the *StreamIdentification* function to obtain the identifier for the respective stream they belong to. Currently, (i) null identifier, (ii) source address and Virtual Local Area Network (VLAN), and (iii) destination address and VLAN are available in the framework as identification functions [IEE17b]. Then, the *SequenceEncoder* module queries *streamSequenceTable* to obtain the latest observed sequence number for that stream using its identifier. If it is not observed before, i.e., for the first frame of the stream, a random sequence number is generated via the *SequenceGeneration* function, and the frame is encoded accordingly. Otherwise, the

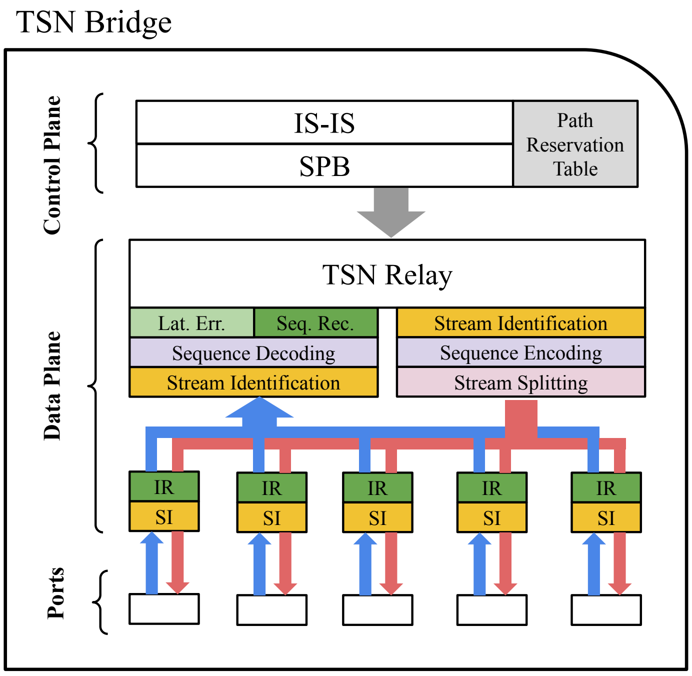


Figure 3.10: The bridge architecture containing all data place functions and control plane protocols. FRER functions are shown in different colors.

existing sequence number is incremented and encoded into the frame. Lastly, encoded frames are split and sent via the configured ports. Stream splitting is not designed as an independent module but as a configuration parameter, *splitFactor*, that specifies the number of distinct paths that the duplicate frames are sent. The configuration of those paths is handled by the control plane functions.

Incoming frames are processed as shown in Figure 3.11. On ingress ports, two types of sequence recovery functions take place: Individual recovery and sequence recovery. In the implementation, *StreamRecovery* module can be configured as one of those functions via *recoveryType* parameter. After identifying the stream, the individual recovery function eliminates duplicates received on a particular port. For the elimination, match or vector recovery functions can be used according to the configuration of *StreamRecovery* module via the parameter *streamRecoveryFunction*. Then, the frames are sent to the stream recovery function, which uses the stream identification functions and the latent error detection module, implemented in *LatentErrorDetection*. This module detects missing or additional frames, i.e., fewer or more duplicates than the expected number.

As stream recovery functions can be placed in both relays and end-points, the rest of the process is node-dependent. After the sequence recovery process, the frame is processed by the relay module, e.g., as any Ethernet frame. On the other hand, the end-host makes the final decision to either accept or drop the frame, e.g., checking if

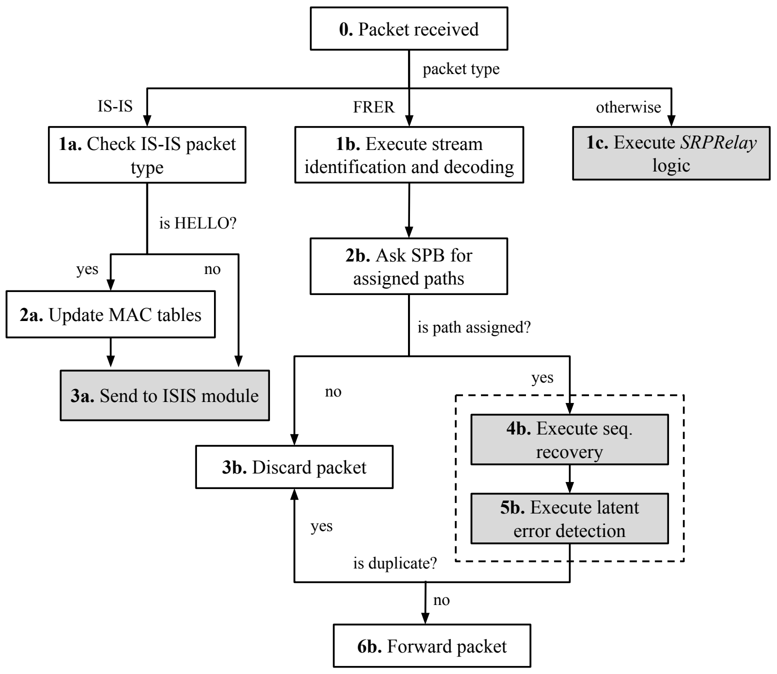

0. Packet received
packet type
IS-IS
FRER
otherwise
1a. Check IS-IS packet type
1b. Execute stream identification and decoding
1c. Execute *SRPRelay* logic
is HELLO?
yes
no
2a. Update MAC tables
2b. Ask SPB for assigned paths
is path assigned?
3a. Send to ISIS module
no
yes
4b. Execute seq. recovery
3b. Discard packet
5b. Execute latent error detection
yes
is duplicate?
no
6b. Forward packet

Figure 3.11: A decision tree for processing incoming frames.

it is the destination nodes or the frame is a duplicate, and thus data plane processing is completed at that point.

Control Plane

The control plane consists of IS-IS and SPB, which are envisioned to be the main path reservation and orchestration mechanisms in TSN.

IS-IS is a distance-vector routing protocol realizing network discovery, topology information sharing and verification as well as routing. In our implementation, each bridge has a unique identifier, *bridgeID*, that can be used instead of MAC addresses to construct paths without keeping track of all MAC addresses for an easier representation. Note that *bridgeID*-MAC address matchings are shared between the bridges during the discovery. This identifier is incremented automatically for each bridge during the initialization. The configuration parameters of IS-IS module are *updatePeriod* and *dbAnnouncePeriod*, which specify the periods for sending *HELLO* and *DB_ANNOUNCE* packets, respectively. *HELLO* packets are used for the discovery of immediate neighbors and each bridge sends its *isis* database via *DB_ANNOUNCE* packets. Those packets are distinguished by the IS-IS Ethernet tag, which is 0x88cc.

When a *HELLO* packet is received, the IS-IS module of the bridge records the source of the packet as its direct neighbor in its database. Similarly, after a database announcement packet is received, the module updates its IS-IS database. Lastly, the module provides Dijkstra's shortest path function to find end-to-end paths between the given source and destination MAC addresses as well as the source and destination *bridgeID* in case of inter-bridge routing.

Utilizing the IS-IS module, the SPB module is responsible to assign multiple paths to a stream between given end-hosts (or practically between given edge bridges). SPB has the *isid* parameter representing the Service Identifier (ISID) that helps to define extended virtualized domains together with Virtual Local Area Network (VLAN) identifiers. Accordingly, IS-IS is modified to construct the paths only among the bridges having the same *isid*, i.e., defining intra-domain paths. It also requires disseminating ISIDs so that a bridge can know with which other bridges it is in the same service group. ISID is thus embedded to the IS-IS *HELLO* packets.

In our implementation, the assignment of paths can be performed in alternative ways. SPB can directly find k shortest paths s.t. $k = splitFactor$ to satisfy the redundancy requirements of FRER. Since *TSNRelay* has the $splitFactor$ as a configuration parameter, it can directly request the required number of paths to split a stream. It is also capable to find k shortest-disjoint paths for the intended use of FRER. The tie-breaking algorithms of SPB help to consistently select the same k paths for each bridge locally. Another approach is the manual configuration, where a network manager can directly provide paths specified by a sequence of *bridgeID*s. A centralized controller, for instance, directly configures SPB module to respond with the given paths for the respective streams. For manual configuration, we provide a specific path encoding notation that could be used within existing network configuration protocols. In both assignment approaches, SPB records the paths in *PathReservationTable* that enables to store multiple paths per FRER stream.

We used this framework for evaluating our previous proposal that is presented in Section 3.3.1, then extended and generalized it for the overall configuration of FRER. The source code is published as open-source at *https://github.com/UHH-ISS/omnet-802.1cb*. Further implementation details related to the internal modules of OMNeT++ can be found in the original paper [EF21a] (Appendix H).

Takeaways

- The IEEE 802.1CB FRER standard does not suggest a configuration for its functions in terms of their interaction, execution order, or parameter setting. Its configuration mostly depends on (i) which TSN component (relay or endpoint) the FRER functions are deployed in and (ii) which features are essential for the specific use cases, e.g., latent error detection might be optional.. Besides, it should still be investigated if its configuration impacts the time-sensitive communication, e.g., inducing further processing delay.
- Although IS-IS and SPB are mentioned in the standards for the management of TSN networks, their seamless integration without consideration of any TSN pa-

rameters is challenging. It is still required to configure those control plane protocols with the related FRER parameters, e.g., split factor, to minimize the potential manual configuration effort. These two takeaways also answer the `RQ.T2.2` by revealing the issues of data and control plane configurations for deploying FRER.

3.4 Security

While IEEE 802.1 TSN protocols ease the design and configuration of MCSs, they might also introduce new attack vectors that broadly impact various mission-critical domains. Accordingly, this section analyzes the potential security threats against several TSN mechanisms and introduces an open-source intrusion detection system (IDS) for the security monitoring to answer the `RQ.T3`.

QoS and reliability issues of IEEE 802.1 TSN standards have been discussed several times, since their main goal is establishing end-to-end latency guarantees. Meanwhile, their security is usually overlooked. There is, in fact, not any work that systematically addresses potential vulnerabilities and attacks towards these standards. Moreover, there is not an off-the-shelf monitoring or intrusion detection mechanism for TSN protocols. Therefore, the following questions arise in addition to the `RQ.T3`:

- `RQ.T3.1`: What are the security threats and attack vectors against different mechanisms of IEEE 802.1 TSN protocols?
- `RQ.T3.2`: How to develop new monitoring and intrusion detection mechanisms against new TSN-specific threats?

Accordingly, the following publication explores and categorizes more than 30 security threats against various TSN mechanisms such as scheduling, configuration, redundancy, and time synchronization[5]:

> Doğanalp Ergenç, C. Brülhart, J. Neumann, L. Krüger, M. Fischer. On the Security of IEEE 802.1 Time-Sensitive Networking. IEEE International Conference on Communications (ICC), Workshop on Time-sensitive and Deterministic Networking, 2021.

Afterwards, the following publication introduces the first network monitoring and intrusion detection system for IEEE 802.1 TSN protocols, `TSNZeek`, to identify the explored security threats and to notice various other anomalies[6]:

> Doğanalp Ergenç, R. Schenderlein, M. Fischer. TSNZeek: An Open-source Intrusion Detection System for IEEE 802.1 Time-sensitive Networking. IFIP Networking Conference, International Workshop on Time-Sensitive and Deterministic Networking (TENSOR), 2023.

[5]In the first publication, the whole contribution belongs to this thesis. All co-authors helped to improve the quality of the paper with their valuable feedback.

[6]In the second publication, the main contributions of this thesis are designing the overall IDS framework, specifying attacks and their respective notices, and conducting the evaluation. The second co-author implemented several parts of `TSNZeek` in the context of a master's thesis. The third co-author helped to improve the quality of the paper with his valuable feedback.

In the remaining of this section, Section 3.4.1 first describes various TSN-specific security threats. Then, Section 3.4.2 presents the design and implementation details of the `TSNZeek`. The respective publications [EBN+21] and [ESF23] are also attached in Appendix I and J.

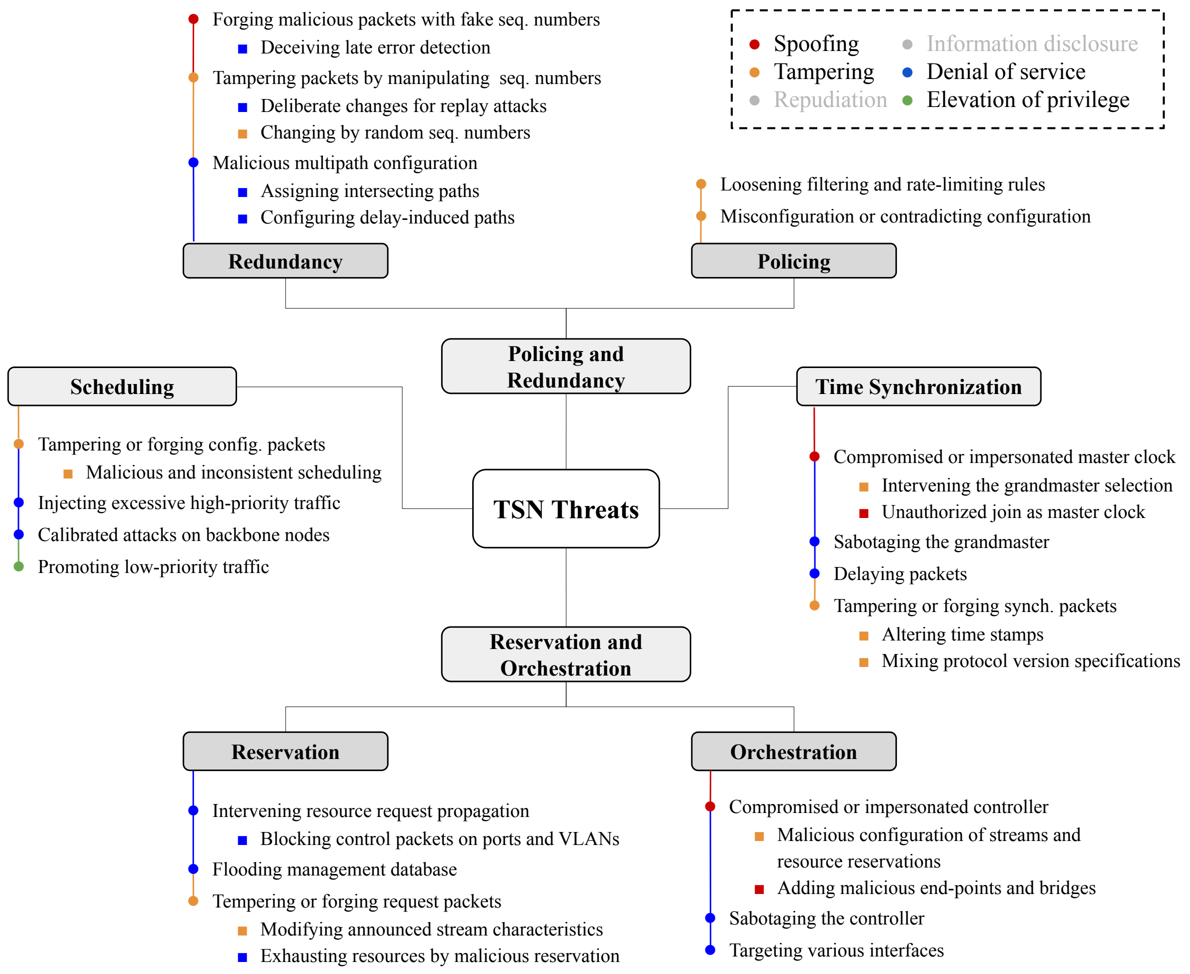
3.4.1 Security Threats of IEEE 802.1 TSN Protocols

The security threats for IEEE 802.1 TSN protocols overlap with several attack vectors of legacy protocols and mechanisms in traditional MCSs. Here, we analyze several TSN threats from the academic literature and industrial reports, and extend those with additional TSN-specific vulnerabilities. Then, following the well-known threat modeling framework, STRIDE [SHLO06], we further categorize them into the main TSN mechanisms. STRIDE covers **S**poofing (red), **T**ampering (orange), **R**epudiation, **I**nformation disclosure, **D**enial of service (blue), and **E**levation of privilege (green) threats against and specifies a guideline for secure system design. Although it is initially proposed for identifying attack vectors against software components, STRIDE also helps to model the primary threats against networks categorically. Figure 3.12 shows a threat tree that illustrates all identified threats categorized into the TSN mechanisms and STRIDE.

The rest of this section briefly describes the threats for the protocols presented in Section 3.1, namely for scheduling, redundancy, and orchestration mechanisms. We also consider those particular threats to develop the intrusion detection system given in Section 3.4.2. More detailed explanations of all threats, i.e., also related to time-synchronization and policing, can be found in the original paper [EBN+21] (Appendix I).

Scheduling Threats

IEEE 802.1Qbv TAS is the primary protocol to establish deterministic communication via scheduling time-sensitive streams (see Section 3.1). Its port-based gate control list can be configured locally or via an external controller, e.g., IEEE 802.1Qcc SRP. The remote configuration can be vulnerable to **packet tampering and forgery on the management protocol** such as Network Configuration Protocol (NETCONF) or Simple Network Management Protocol (SNMP). Even though those protocols can operate on top of security protocols like Secure Shell (SSH) and Transport Layer Security (TLS) [Was11, BLS15], older password-based authentication schemes are still valid and pose security risks. As a result, while an attacker can assign higher-priority and exceeding resources to low-priority traffic, it can also reduce the bandwidth for critical streams, which can lead to missed deadlines due to network congestion.



Spoofing
Tampering
Repudiation
Information disclosure
Denial of service
Elevation of privilege
Forging malicious packets with fake seq. numbers
Deceiving late error detection
Tampering packets by manipulating seq. numbers
Deliberate changes for replay attacks
Changing by random seq. numbers
Malicious multipath configuration
Assigning intersecting paths
Configuring delay-induced paths
Redundancy
Loosening filtering and rate-limiting rules
Misconfiguration or contradicting configuration
Policing
Policing and Redundancy
Scheduling
Tampering or forging config. packets
Malicious and inconsistent scheduling
Injecting excessive high-priority traffic
Calibrated attacks on backbone nodes
Promoting low-priority traffic
TSN Threats
Time Synchronization
Compromised or impersonated master clock
Intervening the grandmaster selection
Unauthorized join as master clock
Sabotaging the grandmaster
Delaying packets
Tampering or forging synch. packets
Altering time stamps
Mixing protocol version specifications
Reservation and Orchestration
Reservation
Intervening resource request propagation
Blocking control packets on ports and VLANs
Flooding management database
Tempering or forging request packets
Modifying announced stream characteristics
Exhausting resources by malicious reservation
Orchestration
Compromised or impersonated controller
Malicious configuration of streams and resource reservations
Adding malicious end-points and bridges
Sabotaging the controller
Targeting various interfaces

Figure 3.12: TSN-specific threats matched with corresponding STRIDE threats.

When an attacker has access to a TSN bridge, there is no mechanism integrated into the scheduling protocols to validate if it satisfies QoS requirements or if no service class is blocked due to congestion. Therefore, **malicious and inconsistent scheduling** cannot be detected.

Moreover, it is possible to abuse the existing gate list configuration without accessing a bridge. For instance, **injecting excessive high-priority traffic** creates high congestion and queueing delays that can cause packet drops and missed deadlines. Similarly, **promotion of low-priority frames to high-priority ones** (assuming that the attacker knows the configured VLAN and priority matchings) can hinder the scheduling scheme.

Lastly, as TSN scheduling enables deterministic end-to-end communication, it is easier to conduct **calibrated attacks on tightly-scheduled nodes** for an attacker [R. 18]. That is, the attacker can identify (i) when high-priority frames are forwarded or (ii) where exactly they are processed at a given. Even though it is not a threat to the scheduling mechanisms per se, it still stems from tight scheduling and deterministic communication.

Control and Orchestration Threats

Attacks towards TSN orchestration and reservation mechanisms, especially towards IEEE 802.1Qcc SRP, can hinder the allocation of required resources in terms of bandwidth and time-scheduling, as well as redundant resources for reliability. As SRP depends on the propagation of stream advertisements, any compromised bridge **intervening in the advertisement propagation** can block forwarding of the respective stream. Similarly, a malicious configuration of a bridge can interrupt the communication in the target domains by **blocking control packets on particular ports or forwarding to certain VLANs**. Instead of an intervention, tampering with the announcement packets to **modify maximum frame size and packet sending frequency** of a stream can mislead other bridges to miscalculate the expected latency, which eventually causes missed deadlines.

Apart from manipulating the existing stream announcements, an attacker can **forge stream reservation requests** as a malicious talker to exhaust available resources. Such malicious requests can threaten the internal resources of bridges by **flooding limited size of management information database**, where the stream registration objects are stored. Flooding attacks are common threats for legacy protocols like the Address Resolution Protocol (ARP), and they are effective in the absence of rate-limiting and filtering on bridges.

The centralized configuration scheme of SRP (see Section 3.1) inherits the characteristics of the SDN architecture [STB19] including its security considerations [ANYG15, DEz+19]. The controller is responsible for configuring almost all TSN protocols, except the time-synchronization mechanism. Therefore, a **a compromised or impersonated controller** can have a disastrous impact on the overall TSN system. While it can result in **malicious misconfiguration of streams, reserved resources, or endpoints**, an attacker can also **sabotage the controller** with more severe denial of service (DoS) attacks. For such scenarios, even though TSN standards do not specify the design of multiple decentralized controllers, redundancy of the controller can help against a single point of failure. Note that even subtle changes in the configuration, for example, rerouting critical streams to more delay-prone paths, can affect the system long-term by degrading the QoS or causing a missing deadline.

Moreover, indirect threats, i.e., not against the orchestration but using orchestration mechanisms, such as **adding new malicious endpoints or bridges**, can enable multi-step propagating attacks [MG20]. A centralized controller also requires different interfaces to enable configuration between the controller, bridges, and endpoints. Consequently, there is an increased attack surface, where an attacker can **target multiple interfaces** between all these TSN components.

Redundancy Threats

The replication and elimination functions of IEEE 802.1CB FRER are vulnerable to security threats. The elimination mechanism of FRER allows a bridge to forward only the first copy of a frame; it should drop the second copy, i.e., replicated for redundancy, when it is received by the same bridge to prevent stuck senders and loops (see Section 3.1 and 3.3.1). An attacker can **forge malicious packets with fake sequence numbers** so that even the original packets with the same sequence numbers are dropped since they would be identified as duplicates. Similarly, even if an attacker sends forged packets later than the original packets, it can **deceive the late error detection mechanism** by disguising a failed link with those additional packets. Apart from such calibrated attacks, **tampering packets by changing their sequence numbers randomly** can easily induce unexpected packet drops and delivery of out-of-order packets to the listener. Although those attacks do not directly interrupt any services, it undermines redundancy and could be only realized when an actual failure occurs.

FRER does not guarantee in-order delivery of multiple member streams. As a result, an attacker can **maliciously configure multiple paths of redundant streams to induce extra delay to the redundant paths and trigger out-of-order packet delivery**. It hinders seamless redundancy in case of failures and causes degradation in QoS. Another issue of path selection is related to the elimination mechanism. While other redundancy mechanisms of Ethernet guarantee the existence of disjoint paths, e.g., through isolated networks or a ring topology [Net16], FRER introduces the elimination function to tackle duplicate packets. As a result, **rerouting the stream through intersecting paths** can cause unexpected packet drops and decrease the intended level of redundancy (see Section 3.3.1).

Takeaways

- The principles of several TSN standards are similar to traditional real-time communication protocols, e.g., Precision Time Protocol (PTP) for time synchronization and NETCONF for remote configuration. Therefore, potential security countermeasures for IEEE 802.1 TSN protocols should include (i) current security mechanisms for existing real-time systems, (ii) built-in security mechanisms of TSN such as stream filtering and policing, and (iii) further monitoring and intrusion detection mechanisms for the novel threats. This also answers the research questions `RQ.T3.1`.

- The identified threats should be analyzed further in the context of particular attacker models. While some attacks are only possible in case of a compromised TSN bridge or controller, others can be conducted by any attacker connected to the network. Therefore, they should be considered according to the domain-specific conditions, e.g., connectivity and accessibility of the target network.

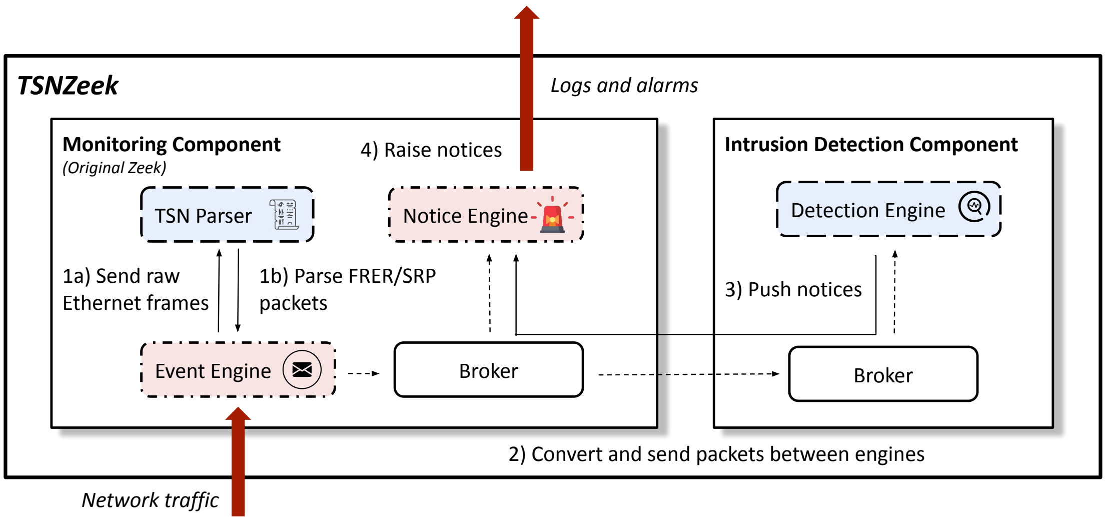


Figure 3.13: The figure shows the overview of `TSNZeek`. The blue/dashed blocks have been implemented from scratch. The red/dotted blocks are existing Zeek modules that we have extended and reconfigured.

3.4.2 Security Monitoring and Intrusion Detection for IEEE 802.1 TSN

While existing security countermeasures can protect MCSs against certain attacks, several TSN-specific threats described in Section 3.4.1 require a security solution that can monitor TSN protocols. This solution should be able to (i) process the new and extended frame structures of the TSN protocols, such as IEEE 802.1 FRER and IEEE 802.1Qbv SRP, and (ii) detect unusual patterns in those protocols. Accordingly, we propose an open-source security monitoring and intrusion detection system, `TSNZeek`, to analyze the new TSN protocols and to detect TSN-specific attacks[7]. It extends Zeek, an a network monitoring solution popularly deployed in real systems and used in academia for research purposes[8].

`TSNZeek` consists of monitoring and intrusion detection components shown in Figure 3.13. The monitoring component processes and log the received TSN traffic. The intrusion detection component obtains the processed frames from the monitoring component and implements the attack recognition logic for TSN-specific attacks. It mainly focuses on threats against FRER and SRP, since they (i) are the critical protocols addressing communication reliability and configuration and (ii) have specific network behavior with new packet types.

The overall operation of `TSNZeek` can be described as follows: the *event engine* distinguishes the incoming packets by their EtherType values, which is a standard header

[7]The source code is available at `https://github.com/UHH-ISS/tsnzeek`.
[8]Zeek Project, https://zeek.org

type of Ethernet frames. SRP and FRER frames are parsed by the *TSN parser* according to the proposed parsing grammar. Those parsed frames are then sent to the *broker* to be translated into Zeek datatypes. It forwards the frames to the *notice engine* and the *Detection Engine*. The former module logs the traffic and specific events using built-in Zeek functionalities, and the latter implements our attack detection rules. The detection engine then pushes notices back to the notice engine when an attack is recognized. The rest of this section briefly presents the capabilities and implementation details of those engines.

Monitoring Component

The monitoring component corresponds to the original Zeek in terms of its working dynamics. It consists of the event engine, the TSN parser, the broker, and the notice engine. The TSN parser (blue, dashed lines in Figure 3.13) has been implemented from scratch. The event and notice engines (red, dashed, and dotted lines) are existing Zeek components that we extended further.

Event Engine: The event engine registers the protocol analyzers during initialization to specify the parsing grammar for respective protocols. Besides, further events are registered in this engine for (i) internal logging facilities and (ii) the broker component to pass incoming packets in a suitable format to other modules. When a registered event is triggered, it calls the respective packet processing function implemented in the TSN parser to parse the whole TSN frame and extract header information.

TSN Parser: This module introduces the functions to parse complex packet structures with many recursive header types. We implemented the parser using *spicy* v1.4.0, which is a grammar generation framework for network protocols and file formats[9]. It strictly follows the definition of frame structure in the respective standards of SRP and FRER to parse any stream that originated from a standard-compliant component. After parsing TSN packets by *spicy* grammar, the extracted data should be converted back to Zeek data types for further processing. Eventually, the parsed packets are sent to the broker to be distributed to all other related components of `TSNZeek`.

Broker: This is the built-in publish/subscribe messaging framework of Zeek. We implement three event topics in the broker: FRER, SRP talker, and SRP listener. Whenever a respective type of frame is received, the broker publishes its content, which is provided by the parser. The notice and detection engines subscribe to those topics and obtain the content of the frames for further analysis accordingly.

Notice Engine: The notice engine flags certain security events and logs received TSN traffic. We used the built-in notice facility of Zeek for this module. We configure

[9]Zeek spicy, https://docs.zeek.org/projects/spicy

the notice engine to log the received FRER and SRP frames partially to avoid an excessive amount of logs. A security event could be an anomaly in the configuration and network behavior, or a detected attack. The detection engine recognizes those events and then respective notice alerts are raised by the notice engine. Those two engines communicate over the broker, and all alerts are published on a specific topic of notice. The complete list of available notices can be found in the original paper [ESF23] (Appendix J).

Intrusion Detection Component

The intrusion detection component consists of the detection engine and another broker to communicate with the monitoring module. It performs traffic analysis to (i) keep the current states of different streams and their configurations, (ii) make per-frame or periodical examinations to detect potential anomalies. Accordingly, it publishes the respective alerts via the broker to be logged by the notice engine.

The detection engine in this component introduces a set of functions to detect various SRP and FRER threats listed in [EBN+21] (Appendix I). These functions are analogous to the rules in a rule-based IDS. Therefore, they are extendable to detect further threats simply as adding new rules. The following lists describe the detection functions together with the attacks they can recognize for SRP and FRER.

- **A1.SRP. Unusual SRP request:** An attacker can send malicious SRP requests to demand a bulky network bandwidth for a stream or register several streams to exhaust available resources. The detection engine detects such scenarios by comparing the requested stream traffic specifications with a predefined threshold value for the maximum bandwidth and frame rates, and raises an alert. It also keeps the rolling average of those values to recognize if an attacker attempts malicious stream reservations whose traffic characteristics significantly differ from the previous ones.
- **A2.SRP. Flooding SRP requests:** An attacker can flood SRP requests to exhaust available resources quickly. The detection engine limits the rate of incoming requests by a predefined threshold and raises an alert for excessive requests.
- **A3.SRP. Changing an existing allocation:** An attacker can forge an SRP request for an already registered stream to reduce its reserved resources to degrade its service quality or increase its reserved resources to exhaust available resources. For each SRP request, it checks if there already exists an active stream and raises alerts. Note that since a legitimate user can also reallocate an active stream, this alert should be considered accordingly.
- **A4.SRP. Dangling resources:** An attacker can reserve network resources to manipulate resource utilization and scheduling without sending any real data traffic

since it can also be detected and filtered by firewalls or network policies. For such cases, the detection engine checks if the reserved resources are in use within a predefined time threshold and triggers a notice.

To detect the listed threats, `TSNZeek` should be able to monitor all SRP traffic, which is forwarded depending on its configuration model, i.e., centralized, hybrid, or distributed (see Section 3.1). While the centralized configuration imposes a centralized `TSNZeek` deployment, a distributed one requires observing the traffic on the edge bridge priorly. Therefore, depending on the targeted threats, multiple `TSNZeek` instances might be required.

- **A5.FRER. Forging fake sequence numbers:** If an attacker can observe the current sequence number of a FRER stream, it can inject malicious frames with that sequence number so that the legit frame would be dropped by the sequence recovery function in TSN bridges. If the attacker injects a frame with the upcoming sequence number, the detection engine triggers a notice when it detects more than one frame with the same sequence number. If the attacker searches for the legit sequence number by sending frames with the random sequence numbers, the detection engine triggers another notice for an out-of-order frame. This also requires monitoring TSN bridges locally to detect where malicious frames are injected and dropped.
- **A6.FRER. Malicious rerouting:** An attacker can subtly reroute the redundant stream through intersecting paths to force the elimination mechanism of FRER (see Section 3.3.1) to drop the duplicate packets and hinder the redundancy. Therefore, the detection engine examines the configured FRER routes, e.g., against malicious misroutings, and intersecting paths. This requires `TSNZeek` to be centrally deployed in collaboration with the network controller that handles the path configuration. It further raises an alert when it detects a missing duplicate packet due to a potential malicious rerouting.
- **A7.FRER. Triggering timeout:** An attacker can enforce FRER functions on TSN bridges for a timeout if it can block all member streams of a FRER stream. This revokes the expected sequence number of the respective stream and enables the attacker to become the valid originator with a forged initial sequence number. The detection engine recognizes the absence of the original member streams by measuring the time passed after the reception of the last frame of the respective streams. It also detects if an active stream has a new sequence number. Both can be detected by only monitoring the edge bridge to which the destination endpoint is attached.

Key results

We evaluated the resource usage of `TSNZeek` and its intrusion detection capabilities in a real TSN testbed. For intrusion detection, the typical evaluation metrics for an IDS, e.g., accuracy, sensitivity [TFM19], are not directly fitting for a rule-based IDS as it can only detect the specific traffic behavior according to the implemented rules. Therefore, we tested the effectiveness of the detection module against the given attacks. As a result, it successfully detects all the listed attacks and raises the respective notices in real-time.

Since IEEE 802.1 TSN protocols perform in the data link layer (Layer 2 in the OSI model), processing the events starting from such low-level communication may easily lead to high resource usage. Accordingly, we measured the CPU utilization of the monitoring and detection components, as well as the packet processing rate and delay of `TSNZeek`. Figure 3.14 shows the CPU usage of the monitoring (Zeek module extended with TSN grammar, red and solid line), the detection (blue and dashed line) components, and also the CPU consumption of Zeek without TSN support (processing non-TSN Ethernet frames, black and dotted line) for an increasing data load from 50 to 250 Mbit/s. When increasing the traffic load, the CPU utilization of the monitoring component increases from 25% at 50Mbit/s to 45% at 250Mbit/s. As shown in the figure, `TSNZeek` consumes only ∼5% more CPU power than the Zeek instance without TSN support. The detection module has a constant resource utilization of around 25% as it only processes singular events sent by the monitoring module.

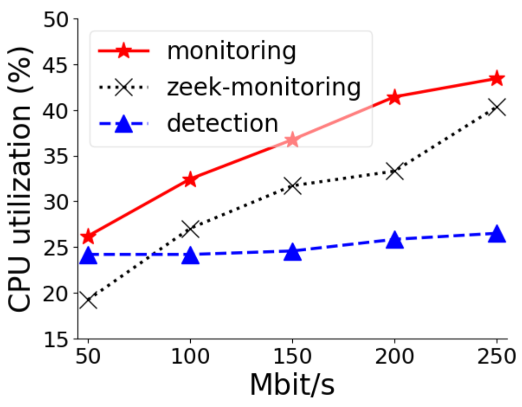


Figure 3.14: CPU utilization of the monitoring and detection components.

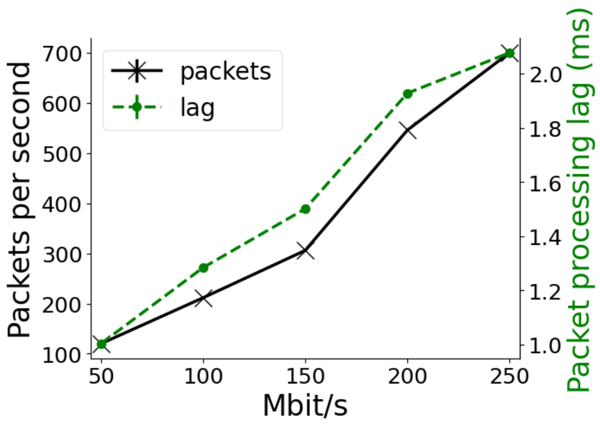


Figure 3.15: Packet processing performance of the monitoring component.

Figure 3.15 shows the packet processing rate and lag that are particularly important for real-time intrusion detection on time-sensitive traffic. The packet processing lag describes the time passed between the reception and parsing of a frame. The figure shows that the packet processing rate (black, solid line) increases proportionally with an increasing data load without packet drops. Increasing load also leads to a higher packet processing lag of up to 2 ms (green, dashed line). For any lag in milliseconds, time-sensitive frames with submillisecond latency requirements may already be delivered before an intrusion alert.

Further details of the implementation, design parameters, testbed setup, and selected attacks can be found in the original paper [ESF23] (Appendix J).

Takeaways

- The effectiveness of an IDS for TSN protocols does not only depend on its detection capabilities, e.g., detection functions and rules, but also its architecture (local, centralized, or distributed), its context awareness (coupling with the information on TSN bridges), and configuration according to the characteristics of the target network. Accordingly, it cannot be only an out-of-the-box solution security solution on the network edge but should be deployed concerning the configuration of TSN protocols. This takeaway also answers the `RQ.T3.2`.
- Critical time-sensitive streams may require sub-millisecond end-to-end latency. Therefore, although it is not critical for a monitoring module, any packet processing lag in the range of a few milliseconds could induce a substantial latency for a real-time intrusion *prevention* (IPS) system. The integration of `TSNZeek` into an IPS thus requires further optimization, potentially shifting the control plane detection functions to the data plane by utilizing network programmability.

Chapter 4
Conclusion

Modern mission-critical systems (MCSs) such as autonomous vehicles, avionics, and industrial networks have evolved from static and closed-loop systems to complex technological ecosystems. They can host a multitude of critical and non-critical services that are interdependent and communicate via different networking technologies. While increasing heterogeneity and connectivity render them more vulnerable to failures and attacks, traditional system design and networking paradigms become incapable of tackling neither stringent QoS nor strict resilience requirements. As a result, the Service-oriented Architecture (SOA) and IEEE 802.1 Time-Sensitive Networking (TSN) are introduced to mission-critical domains to dynamically configure services over virtualized embedded nodes and establish their inter-communication with standardized Ethernet equipment enhanced with time-sensitive capabilities. Those design paradigms enable flexible design and configuration of MCSs, thus implementing novel countermeasures against safety and security threats. However, they also induce additional design complexity and potentially new fault and attack surfaces. Therefore, several research questions have arisen regarding their effective use to design resilient, service-oriented, and time-sensitive MCSs.

Accordingly, in this *cumulative* thesis, we discover the methods of optimal and resilient design of next-generation MCSs focusing on two novel design pillars mentioned above. We first define several research questions regarding the resilience of service-oriented (identified with `RQ.S`) and time-sensitive (identified with `RQ.T`) MCSs. Then, throughout ten research articles, we address those questions aligning with three primary resilience goals, fault tolerance, autonomy, and security. Finally, we present the details and results of those articles under six main contributions (identified with `C`) to resilient service allocation and routing, and resilient time-sensitive networking. Our contributions aligned with the respective research questions are summarized as follows.

Resilient Service Allocation and Routing: Our contributions to resilient service allocation and routing consist of (`C1`) fault-tolerant and optimal joint service allocation and routing for initial service configuration, (`C2`) a distributed and autonomous controlling scheme for service orchestration, and (`C3`) service reconfiguration and data rerouting strategies as proactive security countermeasures.

RQ.S1: **How to deploy inter-connected services within the limited node and link resources by ensuring fault tolerance against potential failures?**

C1. Service-oriented MCSs require to place virtual services on physical network and to establish communication between them. Accordingly, we propose an optimization model, JSAR, to reserve node and link resources for mixed-critical services and their intercommunication demands within the limited resource capacity of MCSs. We further extend it with additional constraints to allocate redundant resources against several potential node and link failure scenarios in [ERF20] (Appendix A). After revealing the complexity of the overall **fault-tolerant** service allocation and routing problem in that study, we propose another model in [ERF21] (Appendix B), JSAR-SP, that shares backup capacity between different services, so that more services and demands can be allocated in the remaining resources. Although JSAR-SP can be solved for larger problem instances than JSAR, we develop several heuristics for finding near-optimal solutions for substantially bigger network and service overlay sizes. As a result, we obtain the optimal initial service distribution for SOA by guaranteeing resilience against all potential single node failures and saving up to 70% of backup resources compared to reserving dedicated backup capacity as in traditional approaches.

RQ.S2: **How to deploy, maintain, and reconfigure mixed-criticality services autonomously with minimal dependency on a controller?**

C2. JSAR and its extensions are rather static models that a centralized controller can compute with its network-wide visibility. However, this controller is a potential single point of failure and may hinder quick and local reactions to failures. Therefore, we propose an **autonomous** and distributed service orchestration framework in [ESF22] (Appendix C) that discards the need for a controller. This framework utilizes the bio-inspired ant colony optimization algorithm that helps to formulate probabilistic task distribution and path finding models. We further adapt this framework with two heuristics, `bia-strict` and `bia-flex`, to enable selecting between better resource utilization or faster computational time for service distribution depending on the system requirements and conditions. Our distributed framework results in near-optimal resource utilization compared to the centralized JSAR.

`RQ.S3`: **How to develop defensive strategies to protect critical services against persistent attacks by leveraging the flexibility of service-oriented architecture?**

`C3.` Although SOA enables dynamically allocating services, a static service configuration can still be a target of well-calibrated and persistent attacks that MCSs are typically threatened by. Accordingly, we propose a spatio-temporal moving target defense framework in [ESKF23] (Appendix D), `PLSCH-MTD`, to dynamically reconfigure services and their data routing in an optimal time-schedule that minimizes the impact of advanced attacks. This framework combines `JSAR` with an attacker-defender game to find feasible service configurations and reconfiguration schedules with a limited defensive budget. `PLSCH-MTD` provides MTD strategies that can protect MCSs up to 90% of their operational time depending on the complexity of potential attack scenarios and the available defensive budget.

Resilient Time-sensitive Networking: The contributions to resilient time-sensitive networking include (`C4`) dynamic and autonomous stream scheduling for initial network configuration, (`C5`) reliable redundant path finding and orchestration for fault-tolerant communication and (`C6`) security monitoring for TSN protocols.

`RQ.T1`: **How to configure mixed-criticality and time-sensitive streams autonomously, satisfying their strict latency requirements?**

`C4.` First, we formulate an optimization model, `TSOR`, for reserving bandwidth and scheduling TSN bridges for mixed-criticality streams. `TSOR` provides the optimal configuration of IEEE 802.1Qbv Enhancements to Traffic Scheduling: Time-Aware Shaper (TAS), which is the most fine-grained scheduling protocol among TSN standards. Then, we utilize `TSOR` in a self-configuration framework in [SBEF21] (Appendix E) to discard end-points from participating stream configuration process. In this framework, TSN bridges autonomously extract the traffic characteristics of incoming data streams, and a centralized controller uses `TSOR` to schedule the bridges and deploy the streams according to their characteristics. The resulting configuration results in a slightly increased end-to-end latency with only 0.25% more delayed frames than the established TSN stream configuration protocol, IEEE 802.1Qcc SRP. Furthermore, we develop several dynamic reconfiguration strategies over `TSOR` in [SBEF22] (Appendix F). They achieve to maintain a similar QoS with varying configuration overhead for increasing communication demands.

RQ.T2: **How to configure redundant communication reliably for time-sensitive streams?**

C5. Beyond their time-sensitive delivery, critical streams in MCSs should not be disrupted by failures. The IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER) protocol enables a flexible configuration of redundant paths for **fault tolerance** and offers several functions for duplicating and eliminating time-sensitive frames. However, this flexibility can also cause unintended frame eliminations due to poorly selected redundant paths that potentially degrade the degree of redundancy. Therefore, we propose the `reassurance` metric in [EF21b] (Appendix G) to select the most reliable redundant paths for an arbitrary degree of redundancy. We then use it to develop a path selection strategy that achieves up to a 40% increase in packet delivery ratio in case of multiple link failures. Besides, we implement an enhanced version of the FRER's elimination function, `eSRF`, to prevent TSN bridges from dropping duplicate frames unintentionally. Lastly, we propose several control plane functions for network discovery and practical configuration of FRER in [EF21a] (Appendix H). To demonstrate the overall framework, we implement these control plane functions and FRER in the OMNeT++ network simulator. This implementation is publicly available at `https://github.com/UHH-ISS/omnet-802.1cb`.

RQ.T3: **How to protect different mission-critical domains against new threats introduced by TSN protocols?**

C6. TSN protocols are envisioned to take place in various critical domains, and thus they are potential targets of advanced attacks towards MCSs. Accordingly, we first identify more than 30 attack vectors against these protocols and systematically categorize them using the STRIDE threat modeling framework in [EBN+21] (Appendix I). However, despite several identified threats against TSN protocols, there is not an established **security** monitoring solution that can analyze TSN traffic. For that, we build the first open-source intrusion detection system in [ESF23] (Appendix J), `TSNZeek`, by extending an existing monitoring tool, Zeek. It has a new packet parser to process TSN protocols and several functions to detect attacks against time-sensitive redundancy and configuration protocols in real-time. `TSNZeek` successfully notices the attacks listed in [EBN+21] with only 5% CPU overhead compared to the original Zeek in a real TSN testbed. The source code of `TSNZeek`, including the new TSN parser and intrusion detection functions, is publicly available at `https://github.com/UHH-ISS/tsnzeek`.

Future Work

Along with our contributions, we discover several other research gaps that remain as future work:

Addressing complexity in coexisting SOA and TSN: Service distribution problems of SOA and scheduling problems of TSN are known to be NP-hard. Therefore, we have developed distinct models to address them separately but not holistically. However, a fully-operating MCS requires to reserve resources for services and their inter-communication (see Section 2.2) and to configure the respective TSN protocols (see Section 3.2 and 3.3) simultaneously, which imposes a joint solution. As a result, system configuration and orchestration procedures (see Section 2.3) should also consider these problems altogether. Therefore, as future work, we will design joint models for a holistic design of SOA and TSN. Here, it requires more comprehensive models than static optimization problems to reflect the complex and interdependent design aspects, e.g., distinct QoS and criticality classes of services and streams, different failure models for nodes and links, etc.

Resilient topology design for MCS: For all contributions of this thesis, we made various assumptions on the topological aspects of MCS due to the lack of available topologies for service-based time-sensitive networks. Therefore, as future work, we will develop topology synthesis models that jointly consider the requirements and constraints of SOA and TSN. These constraints include (i) the connectivity constraints of Ethernet and TSN equipment, (ii) the weight and cost of the overall system, (iii) following domain-specific architectural trends such as zonal design or physical isolation of network parts, and (iv) fault tolerance concerns such as deploying redundant resources and avoiding highly-connected components against a single point of failure. While traditional methods such as linear optimization models can help to synthesize QoS or cost-optimal topologies with limited scalability, more recent approaches like reinforcement learning can more quickly adapt MCS topologies for different service overlays with near-optimal results.

Interconnectivity of TSN systems: The connectivity in MCSs such as industrial networks and smart cars has significantly increased, and at the same time, they become more interconnected with other systems with wireless networking technologies like 5G and WiFi. The emerging 5G technologies are currently being standardized for the use of both within wired TSN and between TSN systems for wireless communication. However, the research and engineering problems arising from the complexity of their joint deployment have not been systematically analyzed yet. For instance, configuring the scheduling, reliability, and management protocols of 5G and TSN holistically for end-to-end communication requires new design and verification models that cover their complex interdependencies. From an engineering standpoint, designing interfaces between the protocol stacks of 5G and TSN, and a common definition of their distinct service classes are essential to build a unified networked system. Therefore, as future work, we will explore arising issues regarding their integration to design resilient hybrid 5G-TSN networks.

Advanced intrusion detection for TSN systems: Traditional MCSs usually have a static network configuration and repetitive communication patterns. It is relatively easier to detect attacks in such setups since a malicious attempt quickly manifests as an anomaly in typical system behavior. However, modern MCSs with increasing heterogeneity and (inter-)connectivity have more complex and dynamic behavior, and a broadened attack surface. Although our intrusion detection system (see Section 3.4) can protect time-sensitive communication against the identified threats, advanced anomaly detection models are still required to recognize internal and external zero-day attacks. Therefore, as future work, we will develop further anomaly-detection models for TSN networks. This requires modeling benign network behavior of time-sensitive networking using state-of-the-art machine learning algorithms, which have already been proven effective against advanced security threats in several domains. Besides, it requires a realistic depiction of TSN-based systems in terms of topology and communication aspects to represent such benign behavior.

Bibliography

[ANYG15] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov. Security in Software Defined Networks: A Survey. *IEEE Communications Surveys Tutorials*, 17(4), 2015.

[BDDK20] O. Burkacky, J. Deichmann, G. Doll, and C. Knochenhauer. *Rethinking Car Software and Electronics Architecture*. McKinsey & Co, 2020. `https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/rethinking-car-software-and-electronics-architecture`.

[BLS15] M. Badra, A. Luchuk, and J. Schoenwaelder. Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication. RFC 7589, 2015.

[Con18] Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT). *Fed4IoT*. 2018. `https://fed4iot.org`.

[Cur20] P.M. Curtis. *An Overview of Reliability and Resiliency in Today's Mission Critical Environment*. 2020.

[DA10] F. Dressler and Ö.B. Akan. Bio-inspired Networking: From Theory to Practice. *IEEE Communications Magazine*, 48(11):176–183, 2010.

[DEz+19] A. Demirpolat, D. Ergenç, E. Öztürk, Y. Ayar, and E. Onur. Software-defined network security. In *Enabling Technologies and Architectures for Next-Generation Networking Capabilities*. IGI Global, 2019.

[DG97] M. Dorigo and L.M. Gambardella. Ant colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[DHM+19] S. Dobson, D. Hutchison, A. Mauthe, A. Schaeffer-Filho, P. Smith, and J.P.G. Sterbenz. Self-Organization and Resilience for Networked Systems: Design Principles and Open Research Issues. *Proceedings of the IEEE*, 107(4):819–834, 2019.

[FAAS+12] D. Fedyk, D. Allan, P. Ashwood-Smith, N. Bragg, J. Farkas, M. Ouellete, M. Seaman, and P. Unbehagen. RFC 6329: IS-IS Extensions Supporting IEEE 802.1aq Shortest Path Bridging. Technical report, Internet Engineering Task Force, 2012.

[FMT+22] T. Fedullo, A. Morato, F. Tramarin, L. Rovati, and S. Vitturi. A Comprehensive Review on Time Sensitive Networks with a Special Focus on Its Applicability to Industrial Smart and Distributed Measurement Systems. *Sensors*, 22(4):1638, 2022.

[GEN20] GENIVI. *Automotive Virtual Platform Specification*. 2020. `https://at.projects.genivi.org/wiki/display/DIRO/Automotive+Virtual+Platform+Specification`.

[IEE16] IEEE 802.1 TSN Task Group. IEEE Standard for Local and Metropolitan Area Networks – Amendment 25: Enhancements for Scheduled Traffic. *IEEE 802.1Qbv-2015*, 2016.

[IEE17a] IEEE 802.1 TSN Task Group. IEEE 802.1 Time-Sensitive Networking (TSN), 2017.

[IEE17b] IEEE 802.1 TSN Task Group. IEEE Standard for Local and Metropolitan Area Networks – Frame Replication and Elimination for Reliability (FRER). *IEEE 802.1CB-2017*, 2017.

[IEE18] IEEE 802.1 TSN Task Group. IEEE Standard for Local and Metropolitan Area Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements. *IEEE Std 802.1Qcc-2018*, 2018.

[JOG+15] S.T. Jones, A.V. Outkin, J.L. Gearhart, J.A. Hobbs, J.D. Siirola, C.A. Phillips, S.J. Verzi, D. Tauritz, S.A. Mulder, and A.B. Naugle. Evaluating Moving Target Defense with PLADD. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2015.

[JOG+17] S.T. Jones, A.V. Outkin, J.L. Gearhart, J.A. Hobbs, J. Siirola, J.D. Daniel, A.P. Cynthia, S.J. Verzi, D. Tauritz, S.A. Mulder, and A.B. Naugle. PLADD: Deterring Attacks on Cyber Systems and Moving Target Defense. In *INFORMS Computing Society Conference*, 2017.

[KJS19] P. Karhula, J. Janak, and H. Schulzrinne. Checkpointing and Migration of IoT Edge Functions. In *2nd International Workshop on Edge Systems, Analytics and Networking*, pages 60–65. ACM, 2019.

[Koe18] A. Koenig. Integrated Sensor Electronics with Self-X Capabilities for Advanced Sensory Systems as a Baseline for Industry 4.0. In *19th ITG/GMA-Symposium Sensors and Measuring Systems*, 2018.

[KW19] J. Klaus-Wagenbrenner. Zonal EE Architecture: Towards a Fully Automotive Ethernet-based Vehicle Infrastructure. In *IEEE Standards Association (IEEE-SA) Ethernet & IP at Automotive Technology Day*, 2019.

[LD07] T.H. Labella and F. Dressler. A Bio-inspired Architecture for Division of Labour in SANETs. In *Advances in Biologically Inspired Information Systems*, pages 209–228. Springer, 2007.

[Mcc76] G.P. Mccormick. Computability of Global Solutions to Factorable Nonconvex Programs: Part I – Convex Underestimating Problems. *Mathematical Programming*, 10(1):147–175, 1976.

[Mer22] Mercedes-Benz. Redundancy for safe conditionally automated driving. `https://group.mercedes-benz.com/innovation/product-innovation/autonomous-driving/redundancy-drive-pilot.html`, 2022.

[MG20] T. Mizrahi and E. Grossman. Deterministic Networking (DetNet) Security Considerations. Internet-Draft draft-ietf-detnet-security-11, Internet Engineering Task Force, 2020. Work in Progress.

[MKK$^+$21] G.M. Makrakis, C. Kolias, G. Kambourakis, C. Rieger, and J. Benjamin. Industrial and Critical Infrastructure Security: Technical Analysis of Real-Life Security Incidents. *IEEE Access*, 9:165295–165325, 2021.

[MP92] K. Makki and N. Pissinou. The Steiner Tree Problem with Minimum Number of Vertices in Graphs. In *2nd Great Lakes Symposium on VLSI*, pages 204–206, 1992.

[MSRL$^+$14] R. Mijumbi, J. Serrat, J. Rubio-Loyola, N. Bouten, F. De Turck, and S. Latré. Dynamic Resource Management in SDN-based Virtualized Networks. In *10th International Conference on Network and Service Management (CNSM)*, pages 412–417, 2014.

[Net16] Industrial Communication Networks. High availability automation networks - Part 3: IEC 62439-3:2016 RLV Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR). 2016.

[NHAM$^+$18] M. Nkomo, G.P. Hancke, A.M. Abu-Mahfouz, S. Sinha, and A.J. Onumanyi. Overlay Virtualized Wireless Sensor Networks for Application in Industrial Internet of Things: A review. *Sensors*, 18(10), 2018.

[OAMAr21] M.N. Olaimat, M. Aizaini Maarof, and B.A.S. Al-rimy. Ransomware Anti-Analysis and Evasion Techniques: A Survey and Research Directions. In *3rd International Cyber Resilience Conference (CRC)*, 2021.

[OSK+19] K. Ogawa, H. Sekine, K. Kanai, K. Nakamura, H. Kanemitsu, J. Katto, and H. Nakazato. Performance Evaluations of IoT Device Virtualization for Efficient Resource Utilization. In *Global IoT Summit (GIoTS)*, 2019.

[PM04] M. Pióro and D. Medhi. *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Elsevier, 2004.

[Pol17] P. Pols. The Unified Kill Chain. *Cyber Security Academy (CSA) Thesis, Hague*, 2017. https://www.unifiedkillchain.com/.

[PPP+18] O.D. Parekh, C.A. Phillips, V. Powers, N. Sakr, and C. Stein. A Scheduling Problem Motivated by Cybersecurity and Adaptive Machine Learning. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2018.

[R. 18] R. Hummen and O. Kleineberg. Cyber-security for Modern TSN Automation Networks. *Industrial Ethernet Book*, 104, 2018.

[Rak12] J. Rak. Fast Service Recovery Under Shared Protection in WDM Networks. *Journal of Lightwave Technology*, 30(1):84–95, 2012.

[SG12] S. Schneele and F. Geyer. Comparison of IEEE AVB and AFDX. In *IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, 2012.

[SHLO06] A. Shostack, S. Hernan, S. Lambert, and T. Ostwald. Uncover Security Design Flaws Using The STRIDE Approach, 2006.

[SL18] W. Sim and J.S. Lee. End-to-end Connectivity Design with Automotive Ethernet & Service-oriented Architecture. In *IEEE Standards Association (IEEE-SA) Ethernet & IP at Automotive Technology Day*, 2018.

[STB19] S.B.H Said, Q.H. Truong, and M. Boc. SDN-Based Configuration Solution for IEEE 802.1 Time Sensitive Networking (TSN). *SIGBED Review*, 16(1), 2019.

[TEK14] M. Thompson, N. Evans, and V. Kisekka. Multiple OS-rotational Environment and Implemented Moving Target Defense. In *7th International Symposium on Resilient Control Systems (ISRCS)*, 2014.

[TFM19] L.N. Tidjon, M. Frappier, and A. Mammar. Intrusion Detection Systems: A Cross-Domain Overview. *IEEE Communications Surveys & Tutorials*, 21(4):3639–3681, 2019.

[TGH+15] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin. Intra-Vehicle Networks: A Review. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):534–545, 2015.

[The20] The Open Group. *Future Airborne Capability Environment (FACE) Technical Standard Edition 3.1*. 2020. `https://www.opengroup.org/face`.

[TMMI16] M. Thompson, M. Mendolla, M. Muggler, and M. Ike. Dynamic Application Rotation Environment for Moving Target Defense. In *Resilience Week (RWS)*, pages 17–26, 2016.

[VH08] A. Varga and R. Hornig. An Overview of the OMNeT++ Simulation Environment. In *1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, 2008.

[VMN21] J. Villaneueva, J. Migge, and N. Navet. QoS-Predictable SOA on TSN: Insights from a Case-Study. In *Automotive Ethernet Congress*, 2021.

[Was11] M. Wasserman. Using the NETCONF Protocol over Secure Shell (SSH). RFC 6242, 2011.

[WHK$^+$19] F. Wilkens, S. Haas, D. Kaaser, P. Kling, and M. Fischer. Towards Efficient Reconstruction of Attacker Lateral Movement. In *14th International Conference on Availability, Reliability and Security (ARES)*, 2019.

[ZAL19] ZAL Center of Applied Aeronautical Research. *Distributed, Extendable, Lightweight, Open, Reliable, Service-oriented Architecture for Next-Gen Mobility (DELIA)*. 2019. `https://delia-project.com`.

[ZXKN17] R.Y. Zhong, X. Xu, E. Klotz, and S.T. Newman. Intelligent Manufacturing in the Context of Industry 4.0: A Review. *Engineering*, 3(5):616 – 630, 2017.

Acronyms

BSRP Backups with Secondary Redundant Path.

JSAR-SP Joint Service Allocation and Routing with Shared Protection.

JSAR Joint Service Allocation and Routing.

MDP Mutually Disjoint Paths.

MLSP Maximum Load to Shortest Path.

PLSCH-MTD PLADD-Scheduling MTD.

PLSCH PLADD-Scheduling.

POBS Post-Optimization Backup Scheme.

RDDP Random Deployment with Disjoint Paths.

SBB Secondary Backup Backbone.

TSOR Time-sensitive Optimal Routing.

eSRF Enhanced Sequence Recovery Function.

ACO Ant Colony Optimization.

ACT Attacker's Capture Time.

AFDX Avionics Full Duplex Ethernet.

ARP Address Resolution Protocol.

BE Best-effort.

CAN Controller Area Network.

DoR Degree of Redundancy.

FRER IEEE 802.1CB Frame Replication and Elimination for Reliability.

GCL Gate Control List.

HSR Highly-available Seamless Redundancy.

IDS Intrusion Detection System.

IPS Intrusion Prevention System.

IS-IS Intermediate System to Intermediate System.

ISID Service Identifier.

LIN Local Interconnected Network.

MAC Media Access Control.

MCS Mission-critical System.

MILP Mixed Integer Linear Programming.

MOST Media Oriented System Transport.

MTD Moving target Defense.

NETCONF Network Configuration Protocol.

PLADD Probabilistic Learning Attacker and Dynamic Defender.

PoSF Probability of Service Failure.

PRP Parallel Redundancy Protocol.

QoS Quality of Service.

SDN Software-defined Networking.

SNMP Simple Network Management Protocol.

SOA Service-oriented Architecture.

SPB IEEE 802.1aq Shortest Path Bridging.

SRF Sequence Recovery Function.

SRP IEEE 802.1Qat Stream Reservation Protocol (SRP) and IEEE 802.1Qcc Enhancements to SRP and Centralization Management.

SSH Secure Shell.

STRIDE Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege.

TAS IEEE 802.1Qbv Enhancements to Traffic Scheduling: Time-Aware Shaper.

TLS Transport Layer Security.

TSN IEEE 802.1 Time-Sensitive Networking.

TT Time-triggered.

VLAN Virtual Local Area Network.

Appendices

Appendix A

Service-based Resilience for Embedded IoT Networks

Abstract

Embedded IoT networks are the backbone of safety-critical systems like smart factories, autonomous vehicles, and airplanes. Therefore, resilience against failures and attacks should be a prior concern already in their design stage. In this study, we introduce a service-based network model as an MILP optimization problem for the efficient deployment of a service overlay to the embedded network by meeting QoS and resilience requirements. We show the complexity and boundaries of the problem and propose several heuristics to relax the service deployment phase and increase the fault-tolerance against node and link failures. Our results indicate that the heuristics achieve results close to the optimum for small sizes of the problem with up to 10^8 time faster solution time. We also show that the heuristics can solve larger problem sizes and can maintain the service availability for 90% of all potential single node failures.

Reference

```
Doğanalp Ergenç, J. Rak, M. Fischer.  Service-Based Resilience for
Embedded IoT Networks.  IEEE/IFIP International Conference on De-
pendable Systems and Networks (DSN), 2020.
```

Contribution

In the forementioned publication, the whole contribution belongs to this thesis. The co-authors helped to improve the quality of the paper with their valuable feedback.

Service-based Resilience for Embedded IoT Networks

Doğanalp Ergenç
Universität Hamburg, DE
ergenc@informatik.uni-hamburg.de

Jacek Rak
Gdansk University of Technology, PL
jrak@pg.edu.pl

Mathias Fischer
Universität Hamburg, DE
mfischer@informatik.uni-hamburg.de

***Abstract*—Embedded IoT networks are the backbone of safety-critical systems like smart factories, autonomous vehicles, and airplanes. Therefore, resilience against failures and attacks should be a prior concern already in their design stage. In this study, we introduce a service-based network model as an MILP optimization problem for the efficient deployment of a service overlay to the embedded network by meeting QoS and resilience requirements. We show the complexity and boundaries of the problem and propose several heuristics to relax the service deployment phase and increase the fault-tolerance against node and link failures. Our results indicate that the heuristics achieve results close to the optimum for small sizes of the problem with up to 10^8 time faster solution time. We also show that the heuristics can solve larger problem sizes and can maintain the service availability for 90% of all potential single node failures.**

***Index Terms*—Resilience, service overlay, optimization, embedded IoT, systems**

I. Introduction

Embedded IoT systems as used in autonomous vehicles, airplanes, and industrial networks have become complex ecosystems. For instance, the latest Tesla autopilot[1] is supported by eight cameras and twelve ultrasonic sensors for high precision and high-quality environmental data. Similarly, with Industry 4.0 intelligent cyber-physical systems emerge that are composed of a multitude of collaborating embedded devices [1], [2]. These devices might also host safety-critical services, where any failure can induce huge damage - in the worst-case up to the loss of human lives.

Moreover, we currently observe that trends from conventional computer networks, like more powerful devices and virtualization, are widely adopted in the (embedded) IoT domain. This can be observed in application scenarios such as modern avionics[2] and smart cities[3]. In both scenarios, there is the need for a non-static and thus flexible deployment of services and functions in embedded systems. As a result, these systems can take over more complex tasks and can operate multiple virtualized services on top of a physical node by maintaining a certain level of process isolation [3], [4]. Furthermore, it allows using node resources more efficiently by starting additional services on demand and even to migrate services in between physical nodes. The increased flexibility of embedded nodes by using virtualization techniques enables dynamic failover schemes like migrating/recovering services after a node failure

[1]Tesla, https://www.tesla.com/autopilot
[2]DELIA, https://delia-project.com
[3]Fed4IoT, https://fed4iot.org

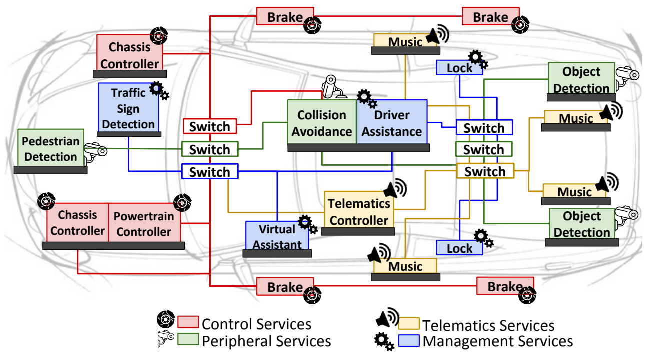


Fig. 1: A simplified reference model for the in-vehicle network consisting of different types of services: Control services (red, wheel icon), peripheral services (green, camera icon), telematics services (yellow, speaker icon), and management services (blue, gear icon).

[5], [6]. Furthermore, as devices become more powerful, more flexible repair and recovery mechanisms during and in the aftermath of failures and attacks can be used, e.g., protection cycles and resilient routing layers [7]. However, all of this comes at the expense of increased complexity, so that these systems become more error-prone and vulnerable. Standard safety concepts like replicating devices and services alone are not sufficient anymore in the presence of sophisticated attackers. Resilience can bring together the safety and security domain by maintaining availability in the presence of failures and attacks, to provide graceful degradation in worst-case, and to recover again [8], [9].

Embedded IoT networks host various potentially interconnected services with specific demands, e.g., communication with bounded delay or recovery from a certain number of node and link failures. Thus, services need to be deployed in the IoT network by considering the capacity and capabilities of nodes and their interconnection. Hence, two degrees of freedom are the result: the service placement on nodes and the routing of data flows in between these nodes.

An example of how we imagine an all Ethernet-based in-vehicle network with virtualized services is given in Fig. 1. Automotive Ethernet will be widely deployed in the future [10]. It allows more flexibility in network configuration and allows the usage of COTS equipment. Therefore, it is a promising solution for evolving in-vehicle networks and the dynamic management of more and more hosted services, e.g., for improved fail-over [11]. In Fig. 1 different types of services are deployed for the reference model: *Control Services* to orchestrate powertrain and chassis domain functions, e.g., brakes and engine control, *Peripheral Services* to collect required data from the environment via (Object and Pedestrian) detectors for collision avoidance, *Telematics Services* for in-car infotainment, e.g., music, and *Management Services* to offer driver assistance managing body functions, e.g., controlling door locks automatically or informing the driver about the traffic signs on the road. Note that, some of the services, e.g., Collision Avoidance and Driver Assistance, are hosted by the same virtualized component.

To realize an automatic braking system, for instance, a group of sensor-connected components host an *object detection service* to detect objects on the road. Another component hosts a *collision avoidance service* that processes information received by the object detection and initiate an emergency brake by informing the component that hosts a *chasis controller service*. In such a service-based architecture service can be easily migrated, e.g., the *collision avoidance service* can be migrated to any other component with sufficient processing power. The *driver assistance service*, as another example, can utilize the components hosting the *music infotainment service* to maintain a *virtual assistant service* that guides the driver vocally in case its actual component hosting it fails. From this point of view, enabling the dynamic service deployment changes the dimension of the resilient communication by benefiting from the flexible design of up-to-date embedded devices [12].

The main contribution of this paper is a model and an optimization problem formulation for the embedding of complex inter-connected services in an IoT network by meeting Quality of Service (QoS) and robustness requirements. Our model results in a fault-tolerant deployment scheme against arbitrary node failures. The resulting system becomes resilient against failures and attacks when it is coupled with an additional dynamic function migration mechanism. We formulate the whole model as a Mixed Integer Linear Program (MILP) to (i) discuss the complexity of the problem that is known to be NP-hard in the literature and (ii) find solutions for the optimum network deployment as a basis for several heuristics we propose additionally. The results indicate that our heuristics achieve results close to the optimum for small sizes of the problem. We also show that the heuristics are getting closer to the optimum when sacrificing resilience. The results also indicate that the heuristics can solve larger problem sizes and allow to embed complex service compositions $10^6 - 10^8$ time faster so that the resulting embedding assures the service availability for 90% of all potential single node failures.

The remaining part of the paper is organized as follows: In Section II, we discuss the related work. Section III introduces the details of our service-based model and optimization problem together with QoS and resilience extensions. In Section IV, we discuss the complexity of the problem. In Section V, we present a heuristic scheme to implement such a model in real systems. We discuss optimization results in different scenarios and constraints in Section VI. Section VII summarizes the paper and proposes future work.

II. Related Work

In this section, we first summarize related work on service allocation and second we describe approaches to enhance the network resilience of embedded IoT networks.

A. Service Allocation

In cloud computing, Software-Defined Networking (SDN), and Network Function Virtualization (NFV) domains, a *service* represents a movable (or relocatable) function that has certain type and characteristics and is allocated to physical nodes. In cloud computing, a service generally provides some specific content, an application, or a platform to users under Service-Level Agreements (SLAs) minimizing the operational costs at the same time. It requires accurate resource orchestration regarding where, when, and how many service instances are deployed [13], [14]. Besides, the dependencies of services to each other [15], service migrations [16], load-balancing [17], and task scheduling [18], are other issues that affect the provider's cost and the user experience.

SDN/NFV services are considered as virtual functions to (i) process and regulate the communication such as firewalls, routers, and load balancers, or (ii) provide network-wide services such as DNS and AAA policies. The proper allocation of a service composition [19], [20] is important to, for instance, minimize operational costs [21] and physical resource fragmentation [22] for the providers, and maximize the service quality [21] and responsiveness [23] for the user experience. Various other studies also address the optimum service allocation and routing problem jointly to deploy the services on the optimal paths [24], [25] to utilize network resources optimally while guaranteeing to satisfy SLAs.

Contrary to the existing works, the service deployment in our study, i.e., in emerging virtualized IoT networks, define the whole communication scheme of an embedded network. That is, as the communication traffic is defined **between** services, inter-service relationships are decisive for network design considering both service deployment and the traffic engineering. In this sense, a service may be a traffic source, destination, or both depending on the system design. Therefore, it is a joint service allocation and inter-service traffic routing problem where routing depends on the service allocation. Moreover, adding resilience requirements to such a dynamic deployment scheme renders the problem even more challenging.

B. Network Resilience

Many traditional approaches leverage graph-related properties of networks to increase their robustness. Against link failures, for instance, finding primary and redundant directed trees [26] as well as multiple disjoint paths [27] are proposed. Some other related studies propose the optimization problems with resilience constraints. In [28], the authors optimize virtual cloud topologies having k redundant instances under network constraints. Similarly, [29] creates survivable virtual groups for each service to guarantee their availability and formulate the deployment of the groups to an underlying network as an optimization problem. Both studies focus on cloud service characteristics as mentioned in Section II-A. In [30], a resource allocation model is proposed for SDN/NFV including fault-tolerance constraints. The authors of [31] consider topology synthesis, routing, and scheduling problems jointly for fault-tolerance in Time-Sensitive Networks (TSN) without including any resource utilization constraint.

In this study, we consider the resilience of services together with optimal resource allocation and routing for inter-service communication. This is in contrast to the state of the art in which the problem is not solved holistically. Other solutions that embody resilience requirements in the optimization formulation only reflect their respective domain characteristics. None of them fits our problem as mentioned in Section II-A.

Furthermore, in conventional safety-critical systems, critical components usually have multiple redundant replicas for failover. From an applicability perspective, our approach allows to maintain this replication factor in the presence of node failures, and thus to tolerate more failures than the replication factor would allow normally. For that, our service-oriented model presumes additional resources as part of the network design.

Table I summarizes the discussion here with a qualitative comparison between our work and the presented studies concerning the criteria discussed in this section. *Resource Efficiency* and *Optimal Routing* represent if a study considers the optimality in resource and network utilization for the service deployment and traffic engineering, respectively. *Resilience* represents if related study proposes any solution for resilience against failures or attacks. Lastly, *Inter-service Dependency* shows whether related study considers the relationship, e.g., hierarchy or communication, between different services as we mentioned its necessity for embedded IoT networks. While none of those studies satisfies all criteria, our work addresses all of them as we show in the rest of the paper.

III. Service-based Model for Embedded Networks

The service-based model aims to embed an overlay network of services into an underlying physical network so that the resulting assignment maintains inter-service data traffic, latency, and foremost resilience demands. In this sense, the service overlay describes a communication scheme between service instances. It can implicitly reflect redundancy for a service, e.g., including multiple instances of the service in a distributed manner. Fig. 2 gives an example for the embedding of a service overlay (black nodes) in the underlying physical network (grey nodes). A service instance can be allocated on a single node to establish communication with other nodes that host neighboring service instances. The overall service and traffic deployment should be restricted by the node and link capacities. Our main goal is to find the optimum allocation of service overlay to the physical network and we present the optimization model in this section. Table II summarizes all terms and definitions of the formulation.

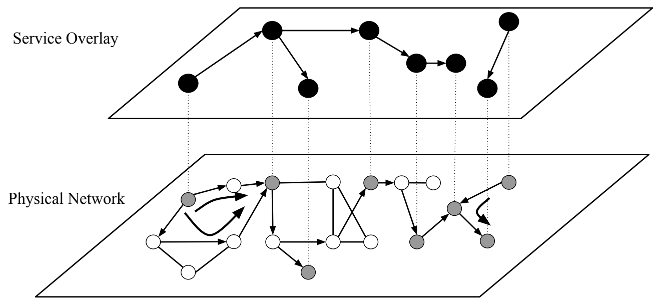


Fig. 2: Service overlay on top of underlay physical network. Dashed lines show how basic service instances are assigned to physical nodes. Grey nodes host the service instances and directed edges show the paths that carry traffic demand.

A. Optimization Problem

A physical network $G(V, E)$ consists of nodes $v \in V$ and links $e \in E$. A service overlay $O(S, D)$ consists of a set of services $s \in S$ and demands $d \in D$. Each service instance $s \in S$ should be deployed to one node to satisfy traffic demands of service overlay. A demand d is defined between a pair of service instances s.t. $\delta : D \mapsto (S\text{x}S)$ and $\delta_d = (s,t) \quad s,t \in S$ to regulate inter-service communication. Besides requiring an amount of data traffic between service instances, a demand also characterizes the QoS and resilience requirements for inter-service communication.

Services consume node resources, e.g., CPU or RAM. A service has a resource demand of τ_s that needs to be provided by a hosting node v. The total resource consumption of node v to host a number of services is limited by its resource capacity r_v as follows:

$$\sum_{s\in S} y_{sv}\tau_s \le r_v \qquad \forall v \in V \tag{1}$$

y_{sv} is a binary variable to decide if s is hosted by v. Since each $s \in S$ should be hosted by a node, the following constraint needs to be satisfied:

$$\sum_{v\in V} y_{sv} \ge 1 \qquad \forall s \in S \tag{2}$$

Inter-service communication is established between source and destination nodes u and v via different paths $p \in P_{uv}$. P_{uv} is the set of all possible paths between those nodes. The set of all paths P is pre-computed and given as input to the problem. To satisfy demand d from s to t s.t. $\delta_d = (s,t)$, we should ensure that (i) any two nodes u and v host services s and t, and (ii) the total data flow through paths $p \in P_{uv}$ should be at least the traffic volume h_d of demand d as:

$$\sum_{u,v\in V} \sum_{p\in P_{uv}} y_{su}y_{tv}x_{dp} \ge h_d$$
$$\forall d \in D, \;\; \delta_d = (s,t), \;\; s,t \in S \tag{3}$$

TABLE I: The comparison of the state-of-the-art studies.

Study	Resource Efficiency	Optimal Routing	Resilience	Inter-service Dependency
Espling et al. [15]	✓			✓
Breitgand et al. [16]	✓			
Pu et al. [17]	✓	✓		
Gawali & Shinde [18]	✓	✓		
[21], [22], [23], [24], [25]	✓	✓		
Medard et al. [26]		✓	✓	
Lee et al. [27]		✓	✓	
Barla et al. [28], Xu et al. [29]	✓	✓	✓	
Beck et al. [30]	✓		✓	
Atallah et al. [31]		✓	✓	

TABLE II: Terms and definitions in the optimization problem. *Base* type contains the fundamental elements of the model. *Constant*s are network- and service-related parameters given as input. *Variable*s represent the parameters to be optimized.

Type	Symbols	Set	Interval	Definition
Base	u, v	V		Nodes in the network
	e	E		Link (edges) between nodes
	s, t	S		Basic services
	d	D		A demand between a pair of services
	p	P_{uv}		An end-to-end path between nodes u and v
	f	F		A failure scenario/state
Constant	τ_s	$\Re^*$	[0,∞]	Resource consumption of s
	h_d	$\Re^*$	[0,∞]	Traffic volume of d
	c_e	$\Re^*$	[0,∞]	Maximum link capacity of e
	r_v	$\Re^*$	[0,∞]	Maximum resource capacity of v
	n_s	Z^*	[0,∞]	Required number of instances for s
	l_d	$\Re^*$	[0,∞]	Latency requirement of d
	l^*_e	$\Re^*$	[0,∞]	Latency in e
	k_{sv}	Z^*	[0,1]	Binary variable to indicate if v is capable to host s
	a_{vf}	Z^*	[0,1]	Binary variable to indicate if v is available in failure state f
	θ_{pf}	Z^*	[0,1]	Binary variable to indicate if p is available in failure state f
Variable	x_{dp}	$\Re^*$	[0,∞]	Flow allocated to path p of demand d
	y_{sv}	Z^*	[0,1]	Binary variable to decide if s is hosted by v
	y_{svf}	Z^*	[0,1]	Binary variable to decide if s is hosted by v in scenario f

x_{dp} is the flow allocated to path p of demand d. A path p consists of a set of links and each link $e \in p$ should have sufficient link capacity to carry the flows on p as:

$$\sum_{d \in D} \sum_{\substack{p \in P, \\ e \in p}} x_{dp} \leq c_e \qquad \forall e \in E \tag{4}$$

c_e is the capacity of e and P is the set of all paths s.t. $P = \bigcup_{u,v \in V} P_{uv}$.

B. Further QoS and Resilience Extensions

Until here, we have presented the main problem as *the deployment of services to physical nodes to satisfy inter-service communication demands under resource and link capacity constraints*. In this section, we extend the problem with the additional QoS and resilience constraints.

Latency requirements. Apart from the traffic volume Constraint (3), a demand d also requires to be satisfied within a bounded delay l_d. Therefore, each demand flow x_{dp} should be allocated to path p that guarantees end-to-end latency less than l_d. The latency in a path depends on the characteristics of each link forming the path. Here, l^*_e is the delay on link e and the end-to-end latency is calculated as sum of the delays in each link. Accordingly, the constraint

$$x_{dp}(\sum_{e \in p} l^*_e \ - l_d) \leq 0 \qquad \forall d \in D, \ \forall p \in P \tag{5}$$

ensures any active path s.t. $x_{dp} > 0$ should be suitable for the corresponding demand. Note that the delay l^*_e on a link is not influenced by the traffic and it is assumed as a constant given as input.

Node capability. Even though assuming that every node is capable to host any service gives significant flexibility for the service deployment, it is not always possible in real systems. Embedded nodes may be equipped with different hardware modules and designed for specific tasks. Besides, a system designer might want to ensure that specific services run on specific nodes, because of QoS and security reasons. Therefore, a binary input parameter k_{sv} to specify *if node v is capable (and permitted) to host service s* is added to the problem. Accordingly, Constraint (2) is extended as,

$$\sum_{v \in V} k_{sv} y_{sv} \geq 1 \qquad \forall s \in S \tag{6}$$

to assign services only to capable nodes.

Single-node demand allocation. Demands are defined between a pair of different services, not necessarily between different physical nodes. Services s and t with demand d s.t. $\delta_d = (s,t)$ can be deployed at the same node. However, the traffic volume h_d is not allocated to any physical path in that case, since it does not require data transfer through the network. Instead of defining extra constraints to enable such a scenario, we modify the definition of a path such as,

$$p = \begin{cases} \{\} & \text{if } p \in P_{uv} \text{ and } u = v \\ \{e_1, e_2 ... e_i\} \;\; e_i \in E & \text{otherwise} \end{cases} \tag{7}$$

This modification introduces *self-paths* that are defined within nodes themselves and do not include physical links s.t. $p = \{\}$. When a flow is allocated to self-paths, it does not affect the link capacity Constraint (2) since none of the links is associated with them. Similarly, latency is omitted in self-paths and it makes a self-path p eligible to be assigned any demand if both services s and t of a demand d are allocated to the same node. Therefore, it does not violate latency and link capacity constraints.

Failure protection. Safety-critical systems should be resilient against different failure scenarios by design. From a modeling perspective, failure state design is a concept where each state represents a failure scenario adding extra constraints to the optimization model [32]. A failure state may include node failures, link failures or both, and each state $f \in F$ is represented by additional input parameters such as indices of failed nodes or links to characterize a failure. Only the initial state i.e., $f = 0$ represents the natural state of a system without any failure. The model reiterates through all states to find an optimum deployment that is resilient against all given failure scenarios. Here, we focus on arbitrary node failures that happen due to an attack, software failure or power cut and affect several service instances on the failed node(s). Such failures may occur in arbitrary nodes independent from the service deployment scheme. Our failure model includes one node failure per state and covers all single node failure scenarios as we define one state for each node in the network. In each failure scenario, a set of binary parameters a_{vf} is given as input to specify if node v is available in failure state $f \in F$ and $\sum_{v \in V} a_{vf} = |N| - 1$ since only one node is assumed to be failed in each state. Note that the model can be extended to consider k-random failures easily by defining further scenarios where each has multiple failed nodes satisfying $\sum_{v \in V} a_{vf} = |N| - k$.

In case of a failure, two main steps should be taken. First, all service instances hosted in the failed nodes must be deployed to other available nodes s.t.,

$$\sum_{v \in V} k_{sv} y_{svf} a_{vf} \geq 1 \qquad \forall s \in S, \; \forall f \in F \tag{8}$$

where y_{svf} represents if service s is deployed to node v in case of failure scenario f. Besides, if v hosts s in any f, there should be a reserved resource in v for s for migration in case of that f occurs. Therefore, Constraint (1) is extended as,

$$\sum_{s \in S} \min(\sum_{f \in F} y_{svf}, 1) \tau_s \leq r_v \qquad \forall v \in V \tag{9}$$

Here, the term with $\min$ function indicates if s deployed to v in any number of states, only τ_s amount of resource need to be occupied in v to activate that service.

Second, routing scheme should be reconsidered since (a) paths may be broken due to failed nodes and (b) the service deployment may change while migrating services in different failure scenarios. Thus, Constraint (3) is extended as,

$$\sum_{u,v \in V} a_{uf} a_{vf} \sum_{p \in P_{uv}} y_{suf} y_{tvf} \theta_{pf} x_{dp} \geq h_d$$
$$\forall d \in D, \;\; \delta_d = (s,t), \;\; s,t \in S, \;\; \forall f \in F \tag{10}$$

where $\theta_{pf} = \prod_{v \in V, v \in p} a_{vf}$ indicates if path p is available in state f, i.e., if all intermediate nodes in p are alive.

Note that the solution constructs a deployment and routing scheme that is resilient to all single node failure scenarios. In this sense, such an approach can be considered as both (i) protective as it reserves required capacity in advance and (ii) restorative as it decides where to migrate services in case of related failure scenario happens. We focus on the single node failure case first to keep the problem size relatively smaller as more complex cases significantly increase the number of possible scenarios, variables, and constraints. However, the model enables us to design such scenarios where multiple failed nodes can be selected arbitrarily, e.g., due to hardware issues, and accidents as well as adjunctly, e.g., connected nodes failed due to a zonal power cut.

C. Objective Function

As we consider bounded-latency as the main QoS metric in our formulation, our objective is the minimization of end-to-end latency. This objective function is decisive to both place services and demands, since the optimal traffic allocation is dependent on the service deployment scheme.

Traffic demands can be forwarded through multiple paths and the whole demand is satisfied when the data on the longest path is received by the destination. In this sense, two end-to-end delay objectives can be considered. *Minimization of the longest active path length* is useful for the type of services that require the complete data to operate. *Maximum amount of data in shortest time* approach is better for the services that do not require to receive the entire data. Equation (11) is formulated for the latter one,

$$\min \quad \sum_{d \in D} \sum_{p \in P} \sum_{e \in p} l_e^* x_{dp} \tag{11}$$

to allocate high traffic demands to the paths with low-delay links.

IV. Complexity Overview

Joint resource allocation and routing combinatorial optimization problems are shown to be NP-hard [33]. The sizes of the physical network and service overlay affect problem complexity in terms of the number of variables and constraints. In this section, we analyze the individual impact of network elements to show the complexity of the problem and develop efficient heuristics presented in Section V. Besides, we show the complexity of constraints and at which cost we linearize them to use profound LP solvers that are already capable to solve complex problems efficiently. Table III shows the new variables introduced after the linearization processes.

Constraint complexity. The traffic demand constraints (3) and (10) include a cubic formulation to ensure that the demand flows are allocated only on the paths between the nodes that host related services. We apply a two-stage linearization process.

a) Linearization of service deployment constraints: The term $y_{su}y_{tv}$ $s,t \in S$, $u,v \in V$ is used to guarantee that required services s and t for demand d s.t. $\delta_d = (s,t)$ are allocated two nodes u and v. The multiplication of two binary variables can be linearized by introducing a new binary variable $w_{stuv} \in W$ under such constraints

$$w_{stuv} \leq y_{su} \tag{12}$$

$$w_{stuv} \leq y_{tv} \tag{13}$$

$$w_{stuv} \geq y_{su} + y_{tv} - 1 \tag{14}$$

for each combination of y_{su} and y_{tv} and it introduces $\mathcal{O}(|S|^2|V|^2)$ new binary variables. However, it is possible to eliminate many w_{stuv} if $\nexists d \in D$ s.t. $\delta_d = (s,t)$. Then, $|W|$ is reducted to $\mathcal{O}(|D|)$.

b) Linearization of flow allocation constraints: In contrast to the service deployment part, this stage includes a binary variable w_{stuv} and a continuous variable x_{dp} instead of two binary variables. A similar linarization process can be applied to define $q^{dp}_{stuv} \in Q$ under those constraints

$$q^{dp}_{stuv} \leq \min(c_e) w_{stuv} \qquad e \in p \tag{15}$$

$$q^{dp}_{stuv} \leq x_{dp} \tag{16}$$

$$q^{dp}_{stuv} \geq x_{dp} - (1 - w_{stuv}) \min(c_e) \qquad e \in p \tag{17}$$

$$q^{dp}_{stuv} \geq 0 \tag{18}$$

where $\min(c_e)$ $e \in p$ is the lowest capacity link in path p and defines the upper bound of x_{dp}. After we filter only matching service-demand pairs, $|Q|$ can be reduced to $\mathcal{O}(|D||P|)$.

Constraint (9) has also a non-linear term, $\min$ function, to distinguish if service s is deployed to v in any failure state. Linearization of $\min$ requires the definition of two binary variables m_{sv} and m^*_{sv} to represent (i) the result of $\min$ function and (ii) the relationship between the parameters of $\min$ function (e.g., which one is bigger than the other), respectively. Accordingly, the following constraints are added

$$1 - \alpha_{svf} \leq |F| m^*_{sv} \tag{19}$$

$$\alpha_{svf} - 1 \leq |F|(1 - m^*_{sv}) \tag{20}$$

$$m_{sv} \leq \alpha_{svf} \tag{21}$$

$$m_{sv} \leq 1 \tag{22}$$

$$m_{sv} \geq \alpha_{svf} - |F|(1 - m^*_{sv}) \tag{23}$$

$$m_{sv} \geq 1 - |F| m^*_{sv} \tag{24}$$

where $\alpha_{svf} = \sum_{f \in F} y_{svf}$ and $|F|$ is the number o failure states. Therefore, additional $2|S||V|$ variables and $6|S||V|$ constraints are introduced. As a result of the linearization process, $\mathcal{O}(|D||P| + |S||V|)$ new variables and constraints are added to the problem.

Impact of the resilience constraints. Each failure state leads to finding an alternative deployment for the specific scenario. Therefore, the number of failure states $|F|$ significantly affects the solution time. Since we limited our design to single node failure scenarios, each state includes exactly one failed node and $|F| = |V|$ is sufficient to address all possible scenarios. As a result, the service deployment Constraint (8) adds $\mathcal{O}(|S||V|)$, the resource capacity Constraint (9) adds $\mathcal{O}(|V|)$, and the traffic Constraint (10) adds $\mathcal{O}(|D||V|)$ constraints. Eventually, $\mathcal{O}(|V|(|D| + |S|))$ extra constraints stem from the resilience design. For number of variables, it costs extra $\mathcal{O}(|S||V|^2)$ service allocation variables, y_{svf}. Besides, the linearization constraints and variables are also multiplied as discussed in the *Constraint complexity*.

Table IV shows how resilience extension affects the problem complexity in terms of the number of constraints and variables, and solution time. Apart from the increasing number of variables and constraints, the solution time significantly increases due to search for a much more restricted deployment for resilience. Table IV shows the impact of resilience constraints in terms of the number of variables, constraints, and solution time.

Impact of the number of links and paths. The number of links $|E|$ and paths $|P|$ are directly correlated. In fact, even one additional link can lead to dozens of alternative paths especially in large networks. While $|P|$ affects (i) the number of variables due to x_{dp} by $\mathcal{O}(|D||P|)$ and (ii) the number of constraints due to Constraint (3) by $\mathcal{O}(|D||P|)$, $|L|$ adds a number of constraints bounded by $\mathcal{O}(|E|)$ due to Constraint (4). Even though it is not possible to eliminate a link without restraining the number of solutions, we can remove a path if none of the demands can be allocated on it due to the latency constraint. The paths that cannot satisfy even the most flexible delay requirement i.e., the demand with the highest delay tolerance cannot be used in the optimal deployment. Since the possible delay on a path is calculatable using link characteristics as in Constraint (5), we used the highest delay requirement as the *cutoff* parameter checking if path delay conforms that parameter. Removing non-conforming ones, the search space for routing is significantly reduced.

Impact of the number of services and demands. Services and demands are the main components to construct a network and their quantities are decisive for the problem complexity. While the number of services $|S|$ multiplies the number of deployment variables y_{sv}, the number of demands $|D|$ determines the number of flow allocation variables x_{dp}. Since $|P|$ is the highest variant in the problem, $|D|$ considerably changes the total number of variables multiplying it. According the results shown in Table V, increasing $|D|$ causes an exponential increase in both number of variables and the solution time in a topology with 20 nodes, average node degree of 3, and $\sim$1600 paths. Note that the most basic scenario without resilience constraints is considered for those experiments. The impact of $|D|$ under resilience constraints is much higher, and shown in

TABLE III: New variables introduced after linearization of the non-linear constraints.

Symbols	Set	Interval	Definition
w_{stuv}	Z^*	[0,1]	Binary variable to specify if s and t are hosted by u and v respectively
q^{dp}_{stuv}	Re^*	[0,1]	Flow allocated to p between u and v to satisfy traffic requirements of d
m_{sv}	Z^*	[0,1]	Binary variable to decide if s is hosted by v in any failure scenario
m^*_{sv}	Z^*	[0,1]	An intermediary binary variable to decide m_{sv}

TABLE IV: Impact of the number of failure states on the number of variables and solution time for the failure-resilient deployment for $|D| = 2$.

Nodes	Non-Resilient			Resilient		
	Variables	Constraints	Solution Time	Variables	Constraints	Solution Time
8	530	1200	0.55s	3008	8181	270.45s
10	3939	7260	0.81s	15947	42434	2710.21s
12	5270	9371	1.13s	28242	77051	9560.57s
14	13751	22170	4.28s	72815	197668	>12h
16	71171	143445	21.53s	439640	1246625	>12h

TABLE V: Impact of the increasing number of demands without resilience constraints on the number of variables and solution time.

Demands	Variables	Solution Time
1	2537	0.55s
3	13037	8.76s
5	27919	32.67s
7	37859	1372.25s
9	50653	17463s

Section VI.

V. Heuristics

Heuristics should cover three dimensions of our problem which are service deployment, inter-service traffic routing, and resilience. To find a complete solution, we first develop greedy heuristics for service deployment and routing since they are directly related. Then, we search for backup nodes and redundant paths to increase resilience against arbitrary node failures. However, even though such a two-stage deployment is quite flexible, it cannot easily optimize networks for all dimensions. Therefore, we design a hybrid heuristic that leverages both a greedy approach and MILP formulation for a better convergence to the optimum solution. However, the involvement of MILP formulation decreases scalability of the heuristic. Table VI summarizes the heuristics including their approaches, i.e., greedy and/or optimized solutions, e.g., for service allocation, routing, and resilience. In the following, we explain these heuristics in detail.

A. Random Deployment with Disjoint Paths (RDDP)

Random Deployment with Disjoint Paths (RDDP) is the greedy heuristic consisting of two phases, service deployment and routing. In the first phase, the services are allocated to a randomly selected pair of nodes that do not host any other service for each demand. If every node hosts at least one service, the active nodes starting from the ones with the highest available resources are selected next. In the second phase, two node-disjoint paths are allocated for the demand between selected nodes, one for the main use and the other one as a redundant backup. If disjoint paths cannot be found between

Algorithm 1: Random Deployment with Disjoint Paths (RDDP)

```
1  $D \leftarrow$ Demands in descending traffic requirements
2  limit $\leftarrow 50$
3  for $demand \in D$ do
4      $services \leftarrow demand$.services
5      $i \leftarrow 0$
6      while $demand$ is not allocated and $i \leq$ limit do
7                                          ▷ First phase: Node selection
8          $i \leftarrow i + 1$
9          $u, v \leftarrow$ Random nodes that can host $services$
10         if Any of $services$ deployed then
11             $u, v \leftarrow$ Pick host node
12         if $u, v$ can host $services$ then
13                                         ▷ Second phase: Path selection
14             $p_1, p_2 \leftarrow$ Shortest disjoint paths from $P_{uv}$
15             if $\exists p_1, p_2 \in P_{uv}$ and $p_1, p_2$ can carry
                 $demand$.traffic then
16                 Allocate $demand$
17                 Update $p_1, p_2$ and $u, v$ capacity
```

those nodes, they are reselected by following the same greedy approach. Algorithm 1 summarizes RDDP.

Note that there is a *limit* parameter in Algorithm 1 to repeat the random node selection process until finding a pair of nodes that can accommodate the services and for which disjoint paths for carrying the respective traffic demand exist. The parameter can be selected according to the network size and the number of demands. *limit* $= 50$ was sufficiently high for all network sizes that we used in our experiments. Therefore, the complexity of RDDP becomes $\mathcal{O}(limit|D|)$. Note that the complexity of finding disjoint paths is excluded as they are given as input parameters.

Even though finding redundant paths increases the resilience of the network traffic, the services on the failed nodes should be still migrated to other nodes. For that RDDP is extended with another greedy heuristic.

TABLE VI: Overview of the proposed heuristics.

Heuristic	Greedy	Optimization	Service Deployment	Routing	Resilience	Scalability
RDDP	✓		✓	✓	∼	✓
RDDP + BSRP	✓		✓	✓	✓	✓
MLSP	✓	✓	✓	✓		
MLSP + POBS	✓	✓	✓	✓	✓	

B. Backups with Secondary Redundant Path (BSRP)

After finding a basis predeployment scheme with RDDP, *Backups with Secondary Redundant Path* (BSRP) heuristic selects a backup node for each service for the migration in case of a node failure. Similar to RDDP, BSRP deploys a backup service instance to the node with the highest available resources in at most $\mathcal{O}(|S||V|)$ iterations. Then, a secondary redundant path, i.e., different from the one offered by RDDP, for each demand between the backup nodes is selected starting from the ones with the lowest cost. This phase induces costs of $\mathcal{O}(|D||P|)$. Algorithm 2 summarizes BSRP.

C. Maximum Load to Shortest Path (MLSP)

As the complexity analysis in Section IV shows the drastic impact of the number of demands on the solution time, we aim to reduce the problem size by allocating some of the demands greedily. *Maximum Load to Shortest Path* (MLSP) heuristic places the most *data-intensive* demands to the shortest, e.g., path with the lowest delay, available paths and deploys related services to end-hosts of those paths, accordingly. If any service is already deployed to a node, one of the shortest paths including that node as the end-host is selected. Then, those services and satisfied demands are excluded from the optimization problem. The rest of the demands, i.e., non-data-intensive ones are again given to the reduced optimization problem under resilience constraints.

Note that the MLSP heuristic includes both greedy service deployment and flow allocation as well as the optimization for different failure scenarios. The services which are placed in advance are assumed to be fixed and their host nodes are not subject to failures. In a real-life scenario, those nodes can be considered as the critical ones whose aliveness is guaranteed by hot-backups. The time complexity of MLSP is $\mathcal{O}(|D||P|)$ as can be derived from Algorithm 3.

For MLSP, we categorize all demands having more than h^* (as a design parameter) traffic requirements as *data-intensive*. It is suitable to optimize the objective function in Equation 11, but such a predeployment can also be performed regarding, e.g., demand priority and criticality.

D. Post-Optimization Backup Scheme (POBS)

After obtaining the optimal deployment with some fixed nodes by MLSP, it is still possible to find a suitable *Post-Optimization Backup Scheme* (POBS) for the fixed nodes and services, given that resources are available for it. Algorithm 4 summarizes the POBS construction. We take the base state, $f = 0$, as the reference deployment where all nodes are alive. Then, we create a new failure scenario for each host node selected by MLSP and move the services on that node to a randomly selected one with sufficient resources. In this case, it is required to reallocate all demands that leverage any of the migrated services, i.e., D_{service}, as well. The demands are moved to the shortest paths between the randomly-selected

Algorithm 2: Backups with Secondary Redundant Path (BSRP)

```
1  $PR \leftarrow$ Basis predeployment scheme
2  $D \leftarrow$ Demands in descending traffic requirements
3  $N \leftarrow$ Nodes in descending available resource capacity
4  for $service$ deployed in $PR$ do
5      ▷ Select backup nodes
6      for $node \in N$ do
7          if $node \neq service$.host and $node$ can host $service$ then
8              $node \leftarrow$ Candidate backup node
9  for $demand \in D$ do
10     ▷ Find redundant paths
11     $services \leftarrow demand$.services
12     $u, v \leftarrow$ Backup nodes for $services$
13     $P_{uv} \leftarrow$ Paths betwen $u$ and $v$ in ascending cost
14     for $p \in P_{uv}$ do
15         if $p$ can carry $demand$.traffic then
16             Update $p$ and $u, v$ capacity
```

Algorithm 3: Maximum Load to Shortest Path (MLSP)

```
1  $D \leftarrow$ Demands in descending traffic requirements
2  $P \leftarrow$ Paths in ascending cost
3  for $demand \in D$ do
4      ▷ Deploy data-intensive demands to nodes
5      $P_{\text{search}} \leftarrow P$
6      if $demand.traffic \geq h^*$ then
7          $services \leftarrow demand$.services
8          if $services$ already deployed then
9              $u, v \leftarrow services$.hosts
10             $P_{\text{search}} \leftarrow P_{uv}$
11         else if Only one service already deployed then
12             $u \leftarrow services$.hosts
13             $P_{\text{search}} \leftarrow P_u$
14     ▷ Find paths for demands
15     for $p \in P_{search}$ do
16         $u, v \leftarrow path$.hosts
17         if $u, v$ can host $services$ and $p$ can carry $demand$ then
18             Allocate $demand$
19             Update $p$ and $u, v$ capacity
```

Algorithm 4: Postoptimization Backup Scheme (POBS)

```
1  $V \leftarrow$ Fixed nodes by MLSP
2  for $node \in V$ do
3      $u \leftarrow$ A random node other than $node$
4      for $service$ hosted by $node$ do
5                                          ▷ Find a backup node
6          if $u$ can host $service$ then
7              for $demand \in D_{service}$ do
8                  $v \leftarrow$ Node hosting other service of $demand$
9                  for $p \in P_{uv}$ do
10                                         ▷ Migrate traffic and service
11                     if $p$ can carry $demand$.traffic then
12                         Update $p$ and $u$ capacity
```

node (by POBS) and the optimally-deployed host (by MILP) to obtain minimum latency.

Even though POBS finds backup nodes and redundant paths for only limited set of services, it is still asymptotically bounded by $\mathcal{O}(|V||P||D|)$ iterations.

VI. Evaluation

In this section, we evaluate our optimization model and compare it to our heuristics. We first describe our experimental setup, our metrics and then present and discuss results from a detailed performance analysis. This includes a discussion of the scalability of the greedy heuristic RDDP+BSRP and the impact of the data-intensiveness threshold h^* on MLSP and MLSP+POBS. Note that since it does not provide service resilience by itself (but resilience against node and link failures via disjoint paths), RDDP is considered together with BSRP as an enhanced heuristic scheme.

A. Experimental Setup

The optimization problem was implemented in CPLEX 12.7.0 and all experiments (including the ones presented in Section IV) were conducted in a server with 64-core Intel Xeon 2.10Ghz CPU and 256GB RAM. Most of the experiments were conducted for different topologies with 10 nodes and 2.5 node average degree to limit the size of the MILP problem. For the model with resilience constraints, the optimizer utilizes all cores and consumes around 120GB RAM in the initial phases of reduction and optimization of the linear problem. CPLEX leverages the branch-and-bound method and the resource usage reduces significantly to around 10GB while CPU utilization remains high, e.g., around 80% of the cores are actively used in the later phases of the optimization. The heuristics, on the other hand, can be solved in a much shorter time (under a second for RDDP and a couple of minutes for MLSP) and thus we do not observe a considerable resource consumption.

Various sample service overlays were generated for the experiments. For n services, $[n-1, 3n/2]$ demands (in which maximum half of them are data-intensive) were defined ensuring that each service communicates at least one other service, i.e., there is no service without a traffic demand. Both topologies and service overlays are random networks where the links/demands are created probabilistically between nodes/services. Table VII presents the intervals in which each parameter is generated with uniform distribution. The convenience of networks, e.g., connectivity of the network, sufficient capacity for demands and services, are confirmed for each generation.

TABLE VII: Intervals for randomly generated parameters.

Parameter	**Interval**
Link capacity (c_e)	[2.0, 4.0]
Node resource capacity (r_v)	[1.0, 3.0]
Service resource consumption (τ_s)	(0.0-2.0]
Demand traffic requirement (h_d)	[1.0, 5.0]
Data-intensiveness threshold (h^*)	[0.5, 2.5]

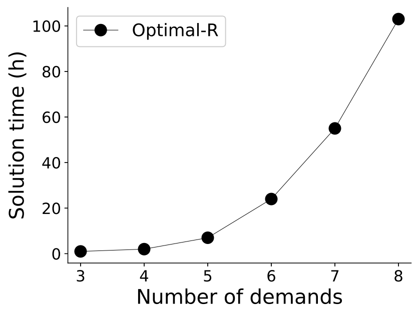
B. Metrics

We measured the solution time, the latency costs, and the probability of service failure. *Solution time* is the time needed to compute solutions for the optimization model and the heuristics. *Latency cost* is defined in Equation 11 and used as the objective function for our model. *Probability of Service Failure* (PoSF) is the ratio of the number of services without backup nodes (due to insufficient node resources) to the number of all services. It represents the percentage of services that fail at most in case of an arbitrary node failure and is used as the resilience metric. While 0% PoSF represents the absolute resilience against any arbitrary node failure, 100% PoSF indicates the failure of all services.

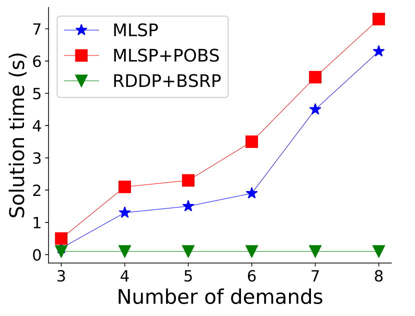
C. Results

a) Solution Time: Fig. 3 shows the solution time for the MILP and heuristics in small topologies with 10 nodes. Fig. 3a shows that the increasing number of demands affect the solution time exponentially between 0.15-100 **hours** in the optimal and single node failure resilient *(Optimal-R)* scenario. In Fig. 3b, on the other hand, sub-optimal solutions are found in **seconds** for the same topologies and service overlays. Since MLSP and MLSP+POBS still solve a reduced MILP, they are slower than RDDP+BSRP. Here, while MPLS+POBS is up to x10^6 times faster than the *Optimal-R*, the greedy approach RDDP+BSRP offers x10^8 times better solution time.

b) Performance Comparison: Even if greedy heuristics are significantly faster than solving the optimization problem, they cannot assure the optimal solution and maximum resilience againt single random node failures. Fig. 4 compares heuristics with the resilient *(Optimal-R)* (i.e., where all services are alive in case of any single node failure) and non-resilient *(Optimal)* (i.e., service deployment and routing without any resilience constraint) solutions in terms of the latency cost and PoSF. As shown in Fig. 4a, the *Optimal* solution has the minimum latency cost and MLSP gets close to the optimum. However, MLSP induces a PoSF of 20-60% as depicted in Fig. 4b, because of the pre-deployed services without backups. After *Optimal-R* (having 0% PoSF), MLSP+POBS



(a) Solution time in hours



(b) Solution time in seconds

Fig. 3: Impact of increasing number of demands on the solution time for a topology with 10 nodes.

offers the best fault-tolerance and improvement from 40% to 5% with increasing number of demands. However, in POBS phase, it cannot find the optimum paths due to random search of the available nodes and becomes 2-4 times more costly in comparison to *Optimum-R*. While RDDP+BRSP is less costly than MLSP+POBS in terms of latency, it is negatively affected by the increasing demands and has a PoSF up to 50%. Note that latency cost and fault-tolerance are directly related to each other. Further fault-tolerance requires to allocate the redundant load to maintain communication in case of failures. In this sense, fault-tolerance costs additional node resources and link capacity.

c) Scalability: Since RDDP+BRSP is the only approach without an optimization stage, it enables us to design larger networks conveniently. Fig. 5a and Fig. 5b show the performance of RDDP+BSRP for increasing demands and nodes, respectively. In Fig. 5a, the PoSF is increasing up to 30% as it is harder to find redundant resources for increasing demands in a fixed-size topology, i.e, 100 nodes. Latency cost is expectedly increasing since the increasing demand should be assigned to the longer paths after shorter ones are priorly utilized.

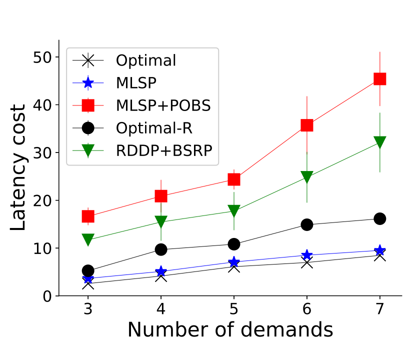
Fig. 5b, on the other hand, reflects the impact of increasing available resources, i.e., the number of nodes and paths, with a fixed number of demands, i.e., 80 demands. When the number of nodes and demands are equal, i.e., 80 nodes and demands, the PoSF can rise up to 50% but it drops to 20% with increasing number of nodes as a consequence of the increased available resources. Being able to deploy more redundant services also results with increasing latency cost as it requires to utilize further paths to ensure communication between redundant services. Note that satisfying a demand already requires the deployment of two services that should also have backup nodes and paths. There should be at least four nodes (and some paths depending on link capacities) per demand to offer such a resilience scheme without a flexible service deployment model. In this sense, RDDP+BSRP offers a scalable and computationally simple solution that can satisfy the resilience requirements to a certain extent.

d) Data-intensiveness Threshold: The efficiency of MLSP and MLSP+POBS depends on the data-intensiveness threshold, h^*. Lower h^* indicates a higher number of greedily deployed demands, a more reduced problem size, and a harder POBS stage. Fig. 6 shows the impact of h^*. As shown in Fig. 6b, MLSP can place more demands for $h^* \leq 1.0$ and it results in a higher number of services without backups and around 40% PoSF which can also go up to 50% for some topologies and service overlays. When it is enhanced by POBS, the PoSF drops to 10% and the deployment becomes fully resilient to single node failures for $h^* > 2.0$. However, 2-3 times lower PoSF leads to a proportional latency cost as seen in Fig. 6a due to the redundant traffic loads allocated for resilience.

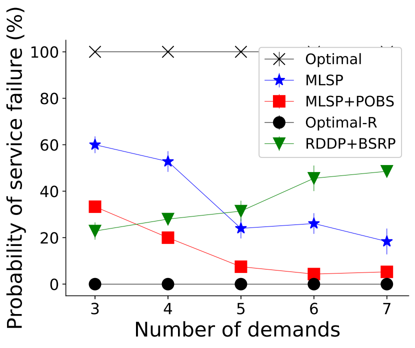
e) Takeaway: As a concrete takeaway, our heuristics show the tradeoff between QoS-optimality, resilience, and scalability. Accordingly, they can be preferred with respect to the desired balance between those design requirements. In the presence of static services, e.g., pre-allocated, non-migratable, or data-intensive, MLSP+POBS can be leveraged to reduce the problem size and can provide a near-optimal solution with a small decrease in resilience, i.e., a slightly higher probability of service failure. However, as it still requires to solve an MILP, it offers only limited scalability. Nevertheless, being a greedy heuristic, RDDP+BSRP offers good scalability at the expense of decreased QoS and resilience. Considering the pros and cons of them, it can be concluded that MLSP+POBS is convenient for relatively small networks having strict QoS requirements and many mission-critical services and communication while RDDP+BSRP can scale to larger networks (in terms of the number of components and services) in which failure-resilience has higher priority than QoS.

VII. Conclusion and Future Work

Embedded IoT networks take over safety-critical tasks and their resilience against failures should be a prior concern already in the design stage. In this study, we presented a service-based network design where the functionalities of a system are defined by inter-service communications, or demands, having certain requirements. We formulated joint service deployment and routing problem as an MILP model and extended it with resilience constraints against random single node failures. The problem is NP-hard, but we also discussed the problem

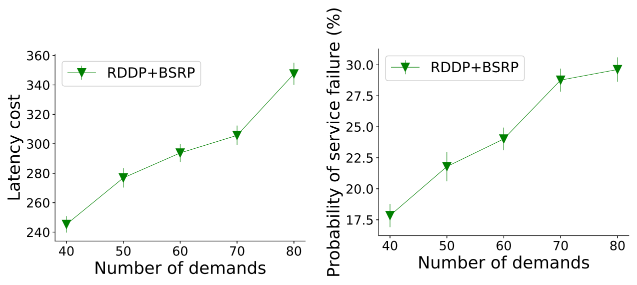


(a) Latency cost

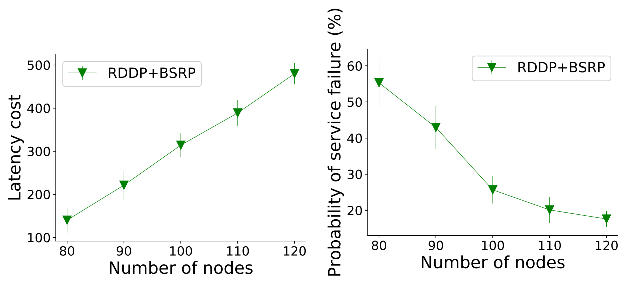


(b) Probability of service failure

Fig. 4: Impact of increasing number of demands for a topology with 10 nodes.

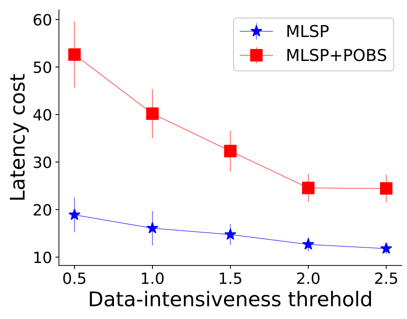


(a) Impact of increasing number of demands for a topology with 100 nodes.

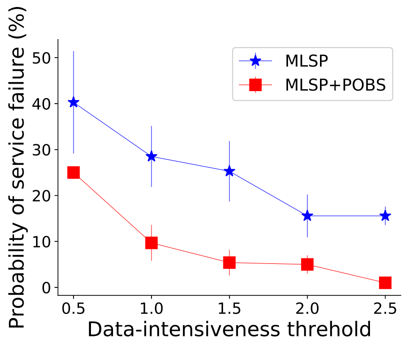


(b) Impact of increasing number of nodes for a service overlay with 80 demands.

Fig. 5: Scalability of RDDP+BSRP for increasing number of demands and nodes.



(a) Latency cost



(b) Probability of service failure

Fig. 6: Impact of increasing h^* for a topology with 10 nodes and a service overlay for 8 demands.

complexity on the basis of the MILP formulation and the resulting scalability issues. As we concluded that the number of demands is the most decisive factor in the solution time, we proposed heuristics to reduce the problem size and find near-optimum greedy solutions. The experiments showed that there is a tradeoff between optimum QoS and resilience and our heuristics promise well-balanced solutions depending on the problem size more than 10^6 times faster than MILP solution.

Considering the problem complexity, we limited the failure scenarios to single node failures. Even though single random failures are frequently used to cover common failure scenarios in networks and distributed systems, more structured massive node failures due to software crashes and vulnerabilities or cyber-attacks are also real threats for embedded IoT networks. Those scenarios will be also considered in our future work in alternative topologies that reflect domain-specific network characteristics.

References

[1] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, “”Intelligent Manufacturing in the Context of Industry 4.0: A Review”,” *Engineering*, vol. 3, no. 5, pp. 616 – 630, 2017.

[2] A. Koenig, "Integrated Sensor Electronics with Self-X Capabilities for Advanced Sensory Systems as a Baseline for Industry 4.0," in *Sensors and Measuring Systems; 19th ITG/GMA-Symposium*, pp. 1–4, June 2018.

[3] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless Sensor Network Virtualization: A Survey," *Communications Surveys and Tutorials, IEEE Communications Society*, vol. 18, pp. 553 – 576, Jan. 2016.

[4] M. Nkomo, G. P. Hancke, A. M. Abu-Mahfouz, S. Sinha, and A. J. Onumanyi, "Overlay virtualized wireless sensor networks for application in industrial Internet of Things: A review," *Sensors (Switzerland)*, vol. 18, no. 10, pp. 1–33, 2018.

[5] K. Ogawa, H. Sekine, K. Kanai, K. Nakamura, H. Kanemitsu, J. Katto, and H. Nakazato, "Performance Evaluations of IoT Device Virtualization for Efficient Resource Utilization," in *2019 Global IoT Summit (GIoTS)*, pp. 1–6, June 2019.

[6] P. Karhula, J. Janak, and H. Schulzrinne, "Checkpointing and migration of iot edge functions," in *Proceedings of the 2Nd International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '19, (New York, NY, USA), pp. 60–65, ACM, 2019.

[7] M. Brinkmeier, M. Fischer, S. Grau, G. Schäfer, and T. Strufe, "Methods for Improving Resilience in Communication Networks and P2P Overlays," *PIK - Praxis der Informationsverarbeitung und Kommunikation*, vol. 32, no. 1, 2009.

[8] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Comput. Netw.*, vol. 54, pp. 1245–1265, June 2010.

[9] J. Rak, *Resilient Routing in Communication Networks*. Springer, 2015.

[10] P. Hank, S. Müller, O. Vermesan, and J. Van Den Keybus, "Automotive ethernet: In-vehicle networking and smart mobility," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1735–1739, March 2013.

[11] S. Tuohy, M. Glavin, E. Jones, M. Trivedi, and L. Kilmartin, "Next generation wired intra-vehicle networks, a review," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 777–782, June 2013.

[12] G. Heiser, "Virtualizing embedded systems - why bother?," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 901–905, June 2011.

[13] J. Chenni Kumaran and M. Aramudhan, "A survey on resource allocation strategies in cloud," *International Journal of Reasoning-based Intelligent Systems*, vol. 10, no. 3-4, pp. 328–336, 2018.

[14] N. K. Pandey, S. Chaudhary, and N. K. Joshi, "Resource allocation strategies used in cloud computing: A critical analysis," *2nd International Conference on Communication, Control and Intelligent Systems, CCIS 2016*, pp. 213–216, 2017.

[15] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth, "Modeling and Placement of Cloud Services with Internal Structure," *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, pp. 429–439, 2016.

[16] D. Breitgand, A. Marashini, and J. Tordsson, "Policy-driven service placement optimization in federated clouds," *IBM Research Division, Tech. Rep*, vol. 9, pp. 11–15, 2011.

[17] L. Pu, L. Jiao, X. Chen, L. Wang, Q. Xie, and J. Xu, "Online resource allocation, content placement and request routing for cost-efficient edge caching in cloud radio access networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1751–1767, 2018.

[18] M. B. Gawali and S. K. Shinde, "Task scheduling and resource allocation in cloud computing using a heuristic approach," *Journal of Cloud Computing*, vol. 7, no. 1, 2018.

[19] X. Li and C. Qian, "A survey of network function placement," *2016 13th IEEE Annual Consumer Communications and Networking Conference, CCNC 2016*, pp. 948–953, 2016.

[20] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A comprehensive survey of Network Function Virtualization," *Computer Networks*, vol. 133, pp. 212–262, 2018.

[21] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," *2015 IEEE 4th International Conference on Cloud Networking, CloudNet 2015*, pp. 171–177, 2015.

[22] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating Virtualized Network Functions," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, 2016.

[23] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On Dynamic service function chain deployment and readjustment," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 543–553, 2017.

[24] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, pp. 98–106, 2015.

[25] G. Lee, M. Kim, S. Choo, S. Pack, and Y. Kim, "Optimal flow distribution in service function chaining," *ACM International Conference Proceeding Series*, vol. 08-10-June-2015, pp. 17–20, 2015.

[26] M. Médard, S. G. Finn, R. A. Barry, and R. G. Gallager, "Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs," *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, pp. 641–652, 1999.

[27] P. P. Lee, V. Misra, and D. Rubenstein, "Distributed algorithms for secure multipath routing in attack-resistant networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, pp. 1490–1501, 2007.

[28] I. B. Barla, D. A. Schupke, M. Hoffmann, and G. Carle, "Optimal design of virtual networks for resilient cloud services," in *2013 9th International Conference on the Design of Reliable Communication Networks (DRCN)*, pp. 218–225, March 2013.

[29] J. Xu, J. Tang, K. Kwiat, W. Zhang, and G. Xue, "Survivable virtual infrastructure mapping in virtualized data centers," in *2012 IEEE Fifth International Conference on Cloud Computing*, pp. 196–203, June 2012.

[30] M. T. Beck, J. F. Botero, and K. Samelin, "Resilient allocation of service function chains," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 128–133, Nov 2016.

[31] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Fault-Resilient Topology Planning and Traffic Configuration for IEEE 802.1Qbv TSN Networks," *2018 IEEE 24th International Symposium on On-Line Testing and Robust System Design, IOLTS 2018*, pp. 151–156, 2018.

[32] M. Pióro and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Elsevier, 2004.

[33] J. Gil Herrera and J. F. Botero, "Resource Allocation in NFV: A Comprehensive Survey," *IEEE Transactions on Network and Service Management*, vol. 13, pp. 518–532, Sep. 2016.

Appendix B

Service-based Resilience via Shared Protection in Mission-critical Embedded Networks

Abstract

Mission-critical networks, which for example can be found in autonomous cars and avionics, are complex systems with a multitude of interconnected embedded nodes and various service demands. Their resilience against failures and attacks is a crucial property and has to be already considered in their design phase. In this paper, we introduce a novel approach for optimal joint service allocation and routing, leveraging virtualized embedded devices and shared backup capacity for the fault-tolerant design of mission-critical networks. This approach operates in phases utilizing multiple optimization models. Furthermore, we propose a new heuristic that ensures resource efficiency and fault-tolerance against single node and link failures as pre-requisite for resilience. Our experiments for different application scenarios indicate that our heuristic achieves results close to the optimum and provides 50% of capacity gain compared to a dedicated capacity protection scheme. Moreover, our heuristic ensures fault-tolerance against at least 90% of all potential single node failures.

Reference

```
Doğanalp Ergenç, J. Rak, M. Fischer.  Service-based Resilience
via Shared Protection in Mission-critical Embedded Networks.  IEEE
Transactions on Network and Service Management (TNSM), Special Issue
on Design and Management of Reliable Communication Networks, 2021.
```

Contribution

In the forementioned publication, the whole contribution belongs to this thesis. The co-authors helped to improve the quality of the paper with their valuable feedback.

Service-based Resilience via Shared Protection in Mission-critical Embedded Networks

Doğanalp Ergenç
Universität Hamburg, DE
ergenc@informatik.uni-hamburg.de

Jacek Rak
Gdansk University of Technology, PL
jrak@pg.edu.pl

Mathias Fischer
Universität Hamburg, DE
mfischer@informatik.uni-hamburg.de

***Abstract*—Mission-critical networks, which for example can be found in autonomous cars and avionics, are complex systems with a multitude of interconnected embedded nodes and various service demands. Their resilience against failures and attacks is a crucial property and has to be already considered in their design phase. In this paper, we introduce a novel approach for optimal joint service allocation and routing, leveraging virtualized embedded devices and shared backup capacity for the fault-tolerant design of mission-critical networks. This approach operates in phases utilizing multiple optimization models. Furthermore, we propose a new heuristic that ensures resource efficiency and fault-tolerance against single node and link failures as pre-requisite for resilience. Our experiments for different application scenarios indicate that our heuristic achieves results close to the optimum and provides 50% of capacity gain compared to a dedicated capacity protection scheme. Moreover, our heuristic ensures fault-tolerance against at least 90% of all potential single node failures.**

***Index Terms*—Mission-critical networks, embedded, resilience, shared protection**

I. Introduction

Mission-critical embedded systems as used in autonomous vehicles, airplanes, and industrial networks have evolved to complex ecosystems. For instance, the latest Tesla autopilot[1] is supported by eight cameras and twelve ultrasonic sensors for high precision and high-quality environmental data. Similarly, together with Industry 4.0 intelligent cyber-physical systems emerged that are composed of a multitude of collaborating embedded devices [1], [2] that run safety-critical services.

Moreover, we currently observe that trends from conventional computer networks, like more powerful devices and virtualization, are widely adopted in the (embedded) IoT domain. McKinsey & Company, for example, considers the virtualization as a key technology to satisfy the latency and reliability requirements of the future autonomous driving [3]. Furthermore, there is ongoing standardization activities such as Automotive Virtual Platform Specification [4] and Future Airborne Capability Environment (FACE) [5] that are preparing the usage of open source virtualization technologies in critical in-vehicle and military systems. As a result and in the future, these systems can host multiple virtualized services on a physical node by maintaining process isolation [6], [7].

From a system design perspective, safety- or mission-critical embedded networks host various potentially interconnected services with specific demands, e.g., certain resource consumption or communication with bounded delay. Virtualization techniques helps to place those services over the physical network and then establish their inter-communication according to their requirements. Besides, as especially the safety-critical services should be protected against any disruption such as attacks or failures, the virtualization enables dynamic failover schemes like migrating/recovering services after node failures [8], [9]. When standard safety concepts like replicating devices and services alone are not sufficient in the presence of sophisticated attackers, such a flexibility in design provides resilience against failures and attacks, maintaining availability with graceful degradation in worst-case or full recovery again.

Eventually, services need to be deployed in the embedded network by considering the capacity and capabilities of nodes and their interconnection. Hence, two degrees of freedom are the result: the service placement on nodes and the routing of data flows in between these nodes. Meanwhile, this configuration should guarantee a certain degree of resilience against the potential malfunctions or threats. To satisfy those requirements, we have modeled the resilient service placement and routing problem addressing single node failures in our preliminary work [10]. Leveraging virtualized embedded devices and virtual services, we have found alternative configurations of the network to reserve required resources for migrating services and flows in case of failures. From this point of view, enabling the dynamic service deployment changes the dimension of the resilient communication by benefiting from the flexible design of up-to-date embedded devices [11]. However, as we also show in [10], the resilient service placement and routing problem is very complex and thus impractical to solve for larger problem sizes.

In this paper, we advance our previous work by utilizing capacity sharing for path protection against single link failures. In comparison to the previous approach that we allocated dedicated backup capacity for traffic demands, we aim to further improve the resource-efficiency by enabling shared capacity use with proper service and demand configurations. We formulate a new optimization scheme extending the one in [10] and then dividing it into multiple steps to reduce its complexity for the embedding of complex inter-connected services by meeting Quality of Service (QoS) and robustness requirements. While our previous model can find the optimal solution for only small-size network, our new multi-step model can scale better ensuring the same degree of fault-tolerance

[1]Tesla, https://www.tesla.com/autopilot

TABLE I: The comparison of the state-of-the-art studies.

Study	Resource Efficiency	Optimal Routing	Resilience	Inter-service Dependency	Shared Protection
Ergenc et al. [10]	✓	✓	✓	✓	
Espling et al. [12]	✓			✓	
Breitgand et al. [13]	✓				
[14], [15], [16], [17], [18], [19], [20]	✓	✓			
Medard et al. [21], Lee et al. [22]		✓	✓		
Barla et al. [23], Xu et al. [24]	✓	✓	✓		
Beck et al. [25]	✓		✓		
Atallah et al. [26]		✓	✓		
[27], [28], [29]		✓	✓		✓
He et al. [30]			✓		✓

against node and link failures. Accordingly, we evaluate the new model in a more realistic topology, which reflects the characteristics of an avionic network architecture. The resulting system becomes resilient against failures and attacks when it is coupled with a dynamic function migration mechanism that realizes the configuration found by our optimization scheme. Concerning our contributions, we

- propose three separate optimization models to solve (i) service placement and routing, (ii) allocation of backup paths with shared capacity use against link failures and (iii) a service migration scheme in case of node failures. While our service allocation and routing model finds the optimal working paths, the shared backup capacity model results in up to 70% capacity gain in comparison to reserving dedicated backup capacity.
- implement the column generation method to enhance our previous model and to solve the extended problem for larger topologies and service overlays more effectively. The resulting model provides fault-tolerance against all single node failures.
- design a new heuristic utilizing Steiner trees to promote shared backup capacity use and improve resource efficiency. Our heuristic results in near-optimal results for the shared backup capacity allocation. It provides more than 90% fault-tolerance against single random node failures that can happen on the host nodes.

The rest of the paper is organized as follows: Section II summarizes related work. In Section III, we introduce our service-based resilience model and the optimization models for the resilient service deployment and routing problem under shared backup capacity. Also, we introduce and explain our heuristic in Section IV. Section V presents our experimental setup and results in detail. Lastly, we summarize our solution and findings in Section VI.

II. Related Work

In this section, we shortly summarize the requirements of our problem. Then, we summarize related work on service allocation, network resilience, and lastly, capacity sharing for backup protection schemes.

Table I shows the requirements for optimal resilient embedded network design that we used as criteria to compare all other studies presented in the rest of this section qualitatively. *Resource Efficiency* and *Optimal Routing* represent the optimality in resource and network utilization for service deployment and traffic engineering, respectively. *Resilience* is one of the main concerns to protect networks against failures or attacks and should be considered for an optimal design of a mission-critical network. *Inter-service Dependency* represents the relationship, e.g., hierarchy or communication, between different services since they are interconnected having specific requirements. Lastly, *Shared Protection* is a concept to use network resources more effectively and is important for the networks that should be more compact and low cost, e.g., for a car or airplane to be lighter and cheaper. In the rest of this section, we discuss the related work according to those criteria.

Service Allocation. In the domains of cloud computing, Software-Defined Networking (SDN), and Network Function Virtualization (NFV), a *service* represents a movable (or relocatable) function of a particular type and characteristics that is allocated to physical nodes. In cloud computing, a service generally provides some specific content, an application, or a platform to users under certain Service-Level Agreements (SLAs) and by minimizing the operational costs at the same time. It requires accurate resource orchestration regarding where, when, and how many service instances are deployed [31], [32]. Besides, the dependencies of services on each other [12], service migrations [13], load-balancing [14], and task scheduling [15] affect the costs of providers and also impact the user experience as well.

SDN/NFV services are considered as virtual functions to process and regulate the communication such as firewalls, routers, and load balancers, or provide network-wide services such as Domain Name System (DNS) and authentication, authorization, and accounting (AAA) services. The proper allocation of those services [33], [34] is important to, for instance, minimize operational costs [16] and physical resource fragmentation [17] for the providers, and maximize the service quality [16] and responsiveness [18] for the user experience. Various other studies address the optimum service allocation and routing problem jointly to deploy the services on the paths [19], [20] to utilize network resources optimally.

Contrary to existing works, the service deployment scheme proposed in this paper focuses on emerging virtualized embedded networks. As the communication traffic is defined between services, inter-service relationships are decisive for network design considering both service deployment and the traffic engineering. Therefore, it is a joint service allocation and inter-service traffic routing problem where routing also depends on the service allocation. Moreover, adding resilience requirements to such a dynamic deployment scheme renders the problem even more challenging.

Network Resilience Many traditional approaches leverage graph-related properties of networks to increase their robustness. Against link failures, for instance, finding primary and redundant directed trees [21] as well as multiple disjoint paths [22] have been proposed. Some other related studies present the optimization problems with resilience constraints. In [23], the authors optimize virtual cloud topologies having k redundant instances under network constraints. Similarly, [24] creates survivable virtual groups for each service to guarantee their availability and formulate the deployment of the groups to an underlying network as an optimization problem. Both studies focus on cloud service characteristics. In [25], a resource allocation model is proposed for SDN/NFV, including fault-tolerance constraints. The authors of [26] consider topology synthesis, routing, and scheduling problems jointly for fault-tolerance in Time-Sensitive Networks (TSN) without including any resource utilization constraint.

In our preliminary work [10], we have considered the resilience of services together with optimal resource allocation and routing for inter-service communication but only for small random networks, i.e., up to ten nodes and seven services. In comparison to other studies here, our new approach can find both the optimal and resilient communication scheme for mission-critical embedded networks in a more resource-efficient manner.

Shared Backup Protection. Shared backup protection increases the resource efficiency in network design as it enables using an amount of capacity mutually between different flows or demands under certain conditions. It is especially prevalent in optical networks where the backup paths can share wavelength links when their working paths are disjoint [35], [36]. In [27], the authors calculate shared backup paths to protect content-connectivity between users and optical datacenters in case of disasters. They propose an ILP and a heuristic to minimize the total number of spectrum slots for optical connections. [28] proposes an efficient shared protection scheme without increasing the length of backups paths compared to the respective working paths and thus promotes fast service recovery. The author formulates the shared protection problem as an ILP for the networks utilizing wavelength-division multiplexing (WDM) and presents heuristics to solve that NP-complete problem. In [30], the authors leverage capacity sharing for survivable virtual network embedding in optical networks to decrease the rejection ratio of an incoming demand due to the lack of resources. They propose a polynomial-time heuristic to calculate shared backup paths. Lastly, [29] utilizes shared protection for a fault-tolerant topology design against more than one failure optimizing capacity usage with two ILP models. Even though all of those studies offer a degree of resilience against node and link failures via shared protection, they assume only static demands that are predefined between certain nodes. Similar to our discussion for the service allocation problems, one of our primary concerns is the deployment of service instances together with working and backup paths where the allocation of demands depends on such deployment and thus is dynamic. Additionally, we have to ensure the resilience of the service deployment, not only the data traffic.

Consequently, in state of the art, the studies do not focus on the problems of service allocation, routing, and resilience jointly in a resource-efficient manner. Those problems are highly relevant for the design of mission-critical networks, which is an avionic network in our use case shown in Section V-A. In that use case, we reflect the domain-specific aspects regarding the topology and service overlay. Adding our shared capacity scheme on top of that, we aim to reduce the resource use and thus the cost of the system. Accordingly, we propose heuristics that solve those problems altogether. From those perspectives, we believe that we propose a solution to a problem that has not been studied holistically yet. Although in our former work [10] we addressed most of those problems, that approach cannot find optimal solutions for networks of reasonable size.

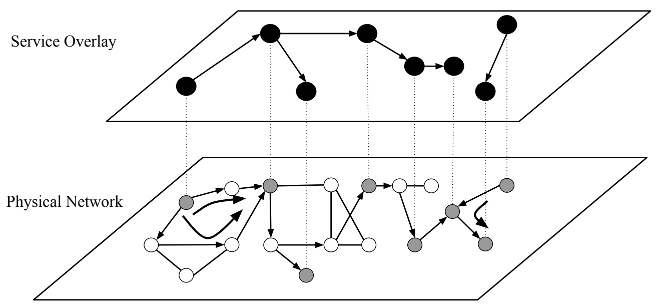


Fig. 1: Service overlay on top of the underlay physical network. Dashed lines show how basic service instances are assigned to physical nodes. Grey nodes host the service instances and directed edges show the paths that carry the traffic demands.

III. Service-based Model for Embedded Networks

Our model aims to embed an overlay network of services into an underlying physical network so that the resulting assignment maintains the inter-service data traffic, latency, and foremost resilience demands. In this sense, the service overlay describes a communication scheme between service instances having certain inter-communication demands. It can also implicitly reflect redundancy for a service, e.g., including multiple service instances in a distributed manner. In this section, we propose an optimization scheme to find the optimal embedding of a service overlay to the physical network. First, we give an overview of the model and our optimization approach that consists of three optimization phases. Then, we explain each phase in more detail.

A. Overview of the Model and the Optimization Scheme

In the service-based model, a service $s \in S$ is defined as a function or virtual instance to be deployed on a physical node $v \in V$. Each s has certain resource requirements τ_s, e.g., CPU or memory, and different criticality levels, e.g., mission-critical or best-effort. Those levels impose a deployment constraint in which only particular nodes can host the service instances with certain criticality, i.e., $k_{sv} = 1$. Another important term, demand $d \in D$, specifies inter-service communication requirements in terms of the end-to-end latency l_d and the amount of data traffic h_d to be exchanged, e.g., required bandwidth. That is, a demand is defined between two service instances and it conditions the data communication in between.

Fig. 1 gives an example for the embedding of a service overlay (black nodes) in the underlying physical network (grey nodes). While a link between two services (an edge between two black nodes) represents a demand, the connection of two physical nodes (an edge between two grey nodes) is a physical link $e \in E$, i.e., having a nominal bandwidth capacity, in the network. A service instance can be allocated on a single node to establish communication with other nodes that host neighboring service instances. The overall deployment should be restricted by (i) the node resource capacities, i.e., r_v for node v, consumed by the services and (ii) link capacities, i.e., c_e for link e, required by the inter-service demands. Besides, the delay induced by path $p \in P$ between two nodes hosting the communicating services imposes a further restriction on the latency on the respective demand.

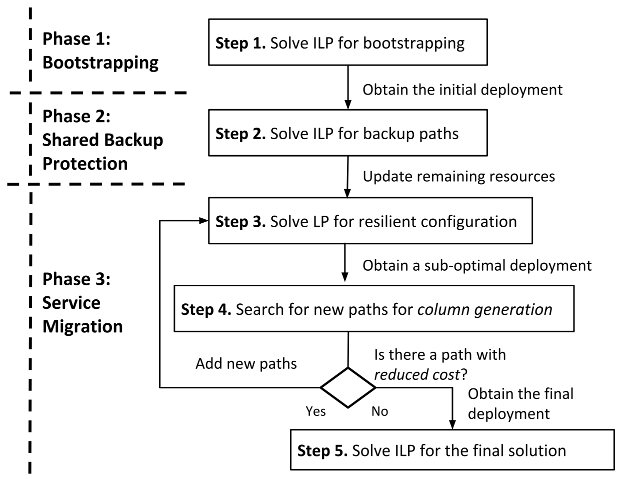

Phase 1: Bootstrapping
Phase 2: Shared Backup Protection
Phase 3: Service Migration
Step 1. Solve ILP for bootstrapping
Obtain the initial deployment
Step 2. Solve ILP for backup paths
Update remaining resources
Step 3. Solve LP for resilient configuration
Obtain a sub-optimal deployment
Step 4. Search for new paths for column generation
Is there a path with reduced cost?
Add new paths
Yes
No
Obtain the final deployment
Step 5. Solve ILP for the final solution

Fig. 2: Multiple steps for the optimal configuration of a resilient virtualized network.

As shown in [10], a single optimization model that reflects optimal service deployment, optimal routing configuration, and fault-tolerance under different failure scenarios results in high complexity. Thus, even for small networks and few services, it might take up to several days to find a configuration with minimum communication latency and a guaranteed resilience against single node failures. For this reason, we split the problem into three phases and different optimization models that need to be solved subsequently as a part of the whole optimization scheme. Those phases that are shown in Fig. 2 are:

1) **Bootstrapping:** In this phase, we find an initial configuration with service deployments and working paths. For that, we formulate an integer linear problem (ILP), namely Bootstrapping-ILP, to find the shortest working paths within the limited node and link resources.
2) **Shared backup protection:** We establish shared backup paths against possible link failures on the working paths found in the phase 1. Using the solution of the previous phase as input, we formulate another ILP, namely Backup-ILP, to minimize the use of backup capacity by maximizing shared protection.
3) **Service migration:** We search for the backup nodes, which communicate with other host nodes with minimum latency, to migrate services in the case of node failures. In this phase, we formulate another optimization model, namely Migration-LP.

Splitting the model into three phases eases the formulation of the constraints. For instance, finding working and backup paths, in phases 1 and 2, could also be considered as a single phase as they are highly dependent. However, it eventually results in complex non-linear constraints, i.e., having complexity higher than quadratic equations. Apart from avoiding such constraints to a certain extent, we also linearize the remaining non-linear constraints to make the overall optimization scheme easily solvable by the existing linear optimization tools. As we solve those linearized models in our experiments, we here introduce the original models as ILP and LP omitting further linearization details. For that, we utilize the McCormick envelopes [37] introducing extra variables and constraints, whose details and complexity are extensively discussed in our preliminary study [10].

Note that splitting the problem into the different optimization phases results in individual optimal solutions for each phase, not a global optimum for the whole scheme involving all constraints at once. We consider this reduction as a trade-off to get a solution which is closer to the optimal one for larger problem instances. Nevertheless, computing the approximation ratio of the split model to a possible singular model requires to formulate and solve such a complex model, which is not practical as we discussed in [10].

In the following subsections, the respective optimization models are presented. Table II briefly summarizes all terms and definitions used in those optimization models.

B. Bootstrapping

In the bootstrapping phase, we solve the optimization model, Bootstrapping-ILP, to find the initial configuration where the service instances and traffic demands are placed on the nodes and working paths. This phase constitutes a base configuration to build further reconfigurations, i.e., service and flow migrations, in the case of node failures and to find backup paths against link failures. The description of all relevant parameters is given in Table II.

TABLE II: Terms and definitions in the optimization problem. *Base* type contains the fundamental elements of the optimization scheme. *Constant*s are network- and service-related parameters given as input. *Variable*s represent the parameters to be optimized.

Type	Symbols	Set	Interval	Definition
Base	u, v	V	-	Nodes in the network
	e	E	-	Link (edges) between nodes
	s, t	S	-	Basic services
	d, q	D	-	A demand between a pair of services
	q	H_d	-	A demand that can share backup capacity with demand d
	p	P_{uv}	-	A path between nodes u and v
	p	$P_{\bar{d}}$	-	A backup path which is disjoint to the working path of d
	f	F	-	A failure scenario
Constants	τ_s	$\Re^*$	[0,∞]	Resource consumption of s
	h_d	$\Re^*$	[0,∞]	Traffic volume of d
	c_e	$\Re^*$	[0,∞]	Maximum link capacity of e
	c_e^*	$\Re^*$	[0,c_e]	Reduced link capacity of e
	r_v	$\Re^*$	[0,∞]	Maximum resource capacity of v
	r_v^*	$\Re^*$	[0,r_v]	Reduced resource capacity of v
	l_d	$\Re^*$	[0,∞]	Latency requirement of d
	l_e^*	$\Re^*$	[0,∞]	Latency in e
	k_{sv}	Z^*	[0,1]	Indicates if v is capable to host s
	a_{vf}	Z^*	[0,1]	Indicates if v is available in failure state f
	θ_{pf}	Z^*	[0,1]	Indicates if p is available in failure state f
	α_{pe}	Z^*	[0,1]	Indicates if p includes e
	x_{dp0}	Z^*	[0,1]	Indicates if a flow is allocated to path p of demand d in a non-failure scenario ($f = 0$)
	y_{sv0}	Z^*	[0,1]	Indicates if s is hosted by v in a non-failure scenario ($f = 0$)
Variables	x_{dp}	Z^*	[0,1]	Decides if d is assigned to p
	x_{dpf}	Z^*	[0,1]	Decides if d is assigned to p in scenario f
	y_{sv}	Z^*	[0,1]	Decides if s is hosted by v
	y_{svf}	Z^*	[0,1]	Decides if s is hosted by v in scenario f
	z_{dp}	Z^*	[0,1]	Decides if d is assigned to backup path p
	z_{de}^*	Z^*	[0,1]	Decides if d is assigned to e
	g_{de}	Z^*	[0,1]	Decides if any demand $q \in H_d$ is assigned to e

Bootstrapping-ILP is given below. x_{dp} and y_{sv} are two binary decision variables to indicate if demand d is assigned to path p and if service s is deployed on node v, respectively.

$$\min \sum_{d \in D} \sum_{p \in P} x_{dp}|p| \tag{1}$$

$$\sum_{s \in S} y_{sv}\tau_s \leq r_v \qquad \forall v \in V \tag{2}$$

$$\sum_{v \in V} k_{sv} y_{sv} = 1 \qquad \forall s \in S \tag{3}$$

$$x_{dp} \leq y_{sv}y_{tu} + y_{tv}y_{su} \qquad \forall d \in D, \forall u, v \in V, \forall p \in P_{uv}, (s,t) \in d \tag{4}$$

$$\sum_{d \in D} \sum_{\substack{p \in P,\\ e \in p}} x_{dp} h_d \leq c_e \qquad \forall e \in E \tag{5}$$

$$\sum_{e \in p} x_{dp} l_e^* \leq l_d \qquad \forall d \in D, \forall p \in P \tag{6}$$

$$\sum_{p \in P} x_{dp} = 1 \qquad \forall d \in D \tag{7}$$

The objective function (1) minimizes the length of selected paths, where $|p|$ represents the path length. Minimizing the total path length can be considered as both performance and cost optimization. That is, allocating shorter paths enables establishing low-latency communications, i.e., here with less hops, and decreasing the number of occupied links, which is especially important for mission-critical networks to reduce the cost and the complexity of the system.

Constraint (2) and (3) ensure that v has sufficient resources to host s and s is deployed on exactly one node that is capable to host s (e.g., equipped with the required hardware). Constraint (4) restricts the flow assignment in a way that d can be deployed on p if only the source and destination nodes u, v of p host required services s and t. Constraint (5) ensures that each link e of p has sufficient resources, e.g., bandwidth, to carry the traffic of d if it is assigned to p. While constraint (6) ensures that p is selected to satisfy the maximum tolerable latency for d, constraint (7) guarantees that d is assigned exactly to one working path. Note that, as inferred in the latest constraint, traffic demands are assumed to be non-bifurcated.

There are two significant uses of the bootstrapping for the next two phases: In the shared protection phase, the bootstrapping eases finding the optimal shared backup paths by providing an initial configuration to (i) detect disjoint working paths that can share backup capacity and (ii) select disjoint backup paths for the given working paths. Formulating mutual disjoint paths, i.e., a working path which is disjoint to both its backup path and other working paths to leverage shared capacity, together with the service allocation results in a complex optimization model (e.g., having cubic constraints). Therefore, computing working paths in advance and then formulating the shared backup protection problem is more convenient to be solved by existing optimization tools. In the service migration phase, the bootstrapping provides a basis of nodes, i.e., host nodes, whose failures are disruptive and result in the unavailability of service instances. Besides, it enables us to keep a part of the initial deployment and to avoid the migration of services and flows on non-failed nodes. Therefore, the bootstrapping phase reduces the search space of the optimization problem, i.e., the number of node failures to consider and the services and flows to reconfigure.

Lastly, in Bootstrapping-ILP, as constraint (4) conforms to both service allocation and routing restrictions, it increases the complexity of the model in $\mathcal{O}(|D||P||S|^2|V|^2)$ in terms of the number constraints. However, practically, only a limited number of such constraints are effective as (i) only the respective services and (ii) only the paths are considered for each demand and each pair of nodes, respectively. Moreover, as it is also a non-linear constraint, the linearization of the multiplication of two binary variables adds extra constraints as well. In terms of variables, due to x_{dp} and y_{sv}, Bootstrapping-ILP is bounded by $\mathcal{O}(|D||P| + |S||V|)$ variables.

C. Shared Backup Protection

In this phase, we formulate another ILP, Backup-ILP, to find backup paths for each demand. Providing communication resilience using dedicated backup paths is generally costly. In particular, if 100% of requested throughput has to be available after a failure, the amount of resources (link capacities) needed to set up the backup path is higher than the resources needed for establishing the working path. To improve the resource efficiency, the concept of sharing the link capacities assigned to backup paths can be applied in such scenarios [38]. In general, sharing the link capacity among several backup paths at a given link is possible if these backup paths protect against different failure scenarios as illustrated in Fig. 3. In the figure, three demands are assigned to the working paths v$_1$-v$_2$-v$_3$, v$_4$-v$_5$-v$_6$, and v$_9$-v$_{10}$-v$_{11}$-v$_6$, and the backup paths v$_1$-v$_7$-v$_8$-v$_3$, v$_4$-v$_1$-v$_7$-v$_8$-v$_6$, and v$_9$-v$_3$-v$_8$-v$_6$, respectively. Accordingly, the shared backup capacities are reserved at links v$_1$-v$_7$ and v$_7$-v$_8$ by the first and second backup paths, at link v$_3$-v$_8$ by the first and third backup paths, and at link v$_6$-v$_8$ by the second and third backup paths for the case of protection against a single node failure as the respective working paths are disjoint and thus subject to different scenarios of single failures. In particular, concerning the scenario of a single link (or a single node) failure covering the majority of failure cases [39], backup path sharing is possible if the respective working paths are mutually link-disjoint (or node-disjoint) [28], [40].

As presented in detail in [28], for a demand d with the requested throughput h_d, the respective backup path at link e in the case of shared protection would require the allocation of (i) no extra capacity if the amount of shareable capacity c^+ already allocated to backup paths at link e is at least h_d or (ii) the extra capacity of $h_d - c^+$ in all other cases. Here, the shareable capacity can be considered as the capacity already reserved for a backup path of another demand q that is accepted earlier and not affected by the same link failure affecting a working path of d. In case $h_q < h_d$, extra capacity $h_d - h_q$ needs to be reserved at the link even though h_q amount

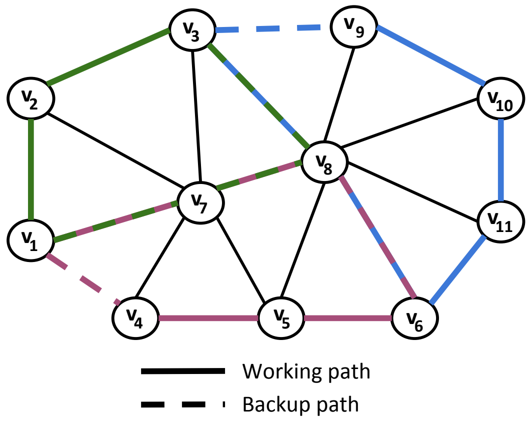


Fig. 3: Example scenario of sharing the backup path capacity.

of capacity can be used by both q and d in case of (different) link failures.

Our model Backup-ILP is given below. Before formulating the problem for a topology G, we update link capacities c_e to c_e^* to denote the capacity not allocated for working paths. Using the initial configuration (i.e., working paths) as the input, we construct a set H_d for each d. It includes demands $\{q_1, q_2 ...\}$ that (i) induce shareable backup capacity with demand d as they have disjoint working paths with d and (ii) have larger traffic demands $h_q > h_d$. As a result, if $q \in H_d$ is assigned on link e, h_d is not necessary to consume extra capacity. The binary decision variables z_{dp} and z^*_{de} represent whether demand d is assigned to path $p \in P_{\bar{d}}$ and link e, respectively. Here, $P_{\bar{d}}$ is a set of disjoint paths to the working path of d obtained from the previous phase and it is computed beforehand. The other decision variable g_{de} shows if any $q \in H_d$ is already assigned to link e. $\bar{g_{de}}$ is the negation of g_{de}.

$$\min \sum_{e\in E} \sum_{d\in D} z^*_{de} \bar{g_{de}} h_d \tag{8}$$

$$\sum_{p\in P_{\bar{d}}} z_{dp} = 1 \qquad \forall d \in D \tag{9}$$

$$z^*_{de} \geq z_{dp} \qquad \forall d \in D, \forall e \in E, p \in P_{\bar{d}} \wedge e \in p \tag{10}$$

$$z^*_{de} \leq \sum_{\substack{p\in P_{\bar{d}},\\ e\in p}} z_{dp} \qquad \forall d \in D, \forall e \in E \tag{11}$$

$$g_{de} \geq z^*_{qe} \qquad \forall d, q \in D, \forall m \in H_d, \forall e \in E \tag{12}$$

$$g_{de} \leq \sum_{q\in H_d} z^*_{qe} \qquad \forall d \in D, \forall e \in E \tag{13}$$

$$\sum_{d\in D} z^*_{de} \bar{g_{de}} h_d \leq c^*_e \qquad \forall e \in E \tag{14}$$

In the model, the objective function (8) minimizes the total shared backup capacity. It aims to increase the resource-efficiency and eventually decrease the design cost of the system by occupying less backup resources.

Constraint (9) ensures exactly one backup path assigned for each demand d. Constraints (10) and (11) configure z^*_{de} for each link e checking if p involving e is a backup path for demand d, i.e., $z_{dp} = 1$. Similarly, constraints (12) and (13) configure g_{de} checking if any demand $q \in H_d$ is assigned to link e. Lastly, constraint (14) ensures that the required resources for d with the highest traffic demand are reserved.

As an exception, if demand d is not suitable to use any amount of shared capacity with another demand, then $H_d = \{\}$. In this case, $g_{de} = 0$ holds strictly and the full amount of traffic for demand d should be assigned to the respective links without considering any shared capacity. Constraint (14) implicitly considers this scenario as well.

Lastly, in terms of complexity, Backup-ILP is bounded by $\mathcal{O}(|D||E||P|)$ constraints and $\mathcal{O}(|D|(|E| + |P|))$ variables including linearized the non-linear constraint, which contains the multiplication of z^*_{de} and g_{de}.

D. Service Migration

Backup paths protect traffic demands against the failures in the intermediate nodes and links of the respective working paths. However, the failure of one of the end nodes hosting a service still disrupts the services and demands. In the last phase, we formulate Migration-LP to find alternative nodes for each service hosted by failed nodes. Defining failure scenarios $f \in F/\{0\}$, we consider the failure case of each node that hosts a service according to the given bootstrapping configuration.

Before formulating the problem for the topology G, we update node and link capacities according to the deployments in the previous phases. In Migration-LP, x_{dpf} and y_{svf} are the binary decision variables that represent whether demand d is assigned on path p and if service s is deployed on node v in the failure scenario f, respectively. Each scenario f is represented by a vector of binary variables a_{vf} that specficies whether node v is not failed in scenario f. Services can be hosted at node v only if $a_{vf} = 1$. θ_{pf} is another binary variable that represents if path p is not broken, i.e., is usable, in scenario f and is decided by the availability of the nodes on p s.t $\theta_{pf} = \prod_{v \in V, \, v \in p} a_{vf}$. According to the service configuration from the bootstrapping phase, the failure scenarios are defined in such a way that each one represents the failure of a single host node, i.e., $\sum_{v \in V} a_{vf} = 1$. Therefore, the number of scenarios $|F|$ equals to the number of distinct nodes hosting services in the initial configuration. Eventually, the resulting configuration of Migration-LP gives an alternative deployment to update the bootstrapping configuration in the case of the respective failure scenarios.

Migration-LP is given below. The objective function (15) minimizes the length of selected paths. Constraints (16)-(21) resemble to constraints (2)-(7) in the bootstrapping phase. Constraints (22) and (23) ensure that if the initial flow assignment and service deployment are not affected by the failure in scenario f, their configuration is kept to avoid the unnecessary reconfiguration of the network. Here, x_{dp0} and y_{sv0} are given as input according to the initial configuration from the bootstrapping phase.

$$\min \sum_{d \in D} \sum_{p \in P} \sum_{f \in F} x_{dpf} |p| \tag{15}$$

$$\sum_{s \in S} y_{svf} \tau_s \le r^*_v \qquad \forall v \in V, \ \forall f \in F \tag{16}$$

$$\sum_{v \in V} k_{sv} y_{svf} a_{vf} = 1 \qquad \forall s \in S, \ \forall f \in F \tag{17}$$

$$x_{dpf} \le \theta_{pf} \left[y_{suf} y_{tvf} + y_{tvf} y_{suf} \right] \quad \forall d \in D, \forall u, v \in V, \forall f \in F \quad \forall p \in P_{uv}, (s,t) \in d \tag{18}$$

$$\sum_{d \in D} \sum_{p \in P} x_{dpf} \theta_{pf} \alpha_{pe} h_d \le c_e \qquad \forall e \in E, \forall f \in F \tag{19}$$

$$\sum_{e \in E} x_{dpf} \alpha_{pe} l^*_e \le l_d \qquad \forall d \in D, \forall p \in P, \forall f \in F \tag{20}$$

$$\sum_{p \in P} x_{dpf} \theta_{pf} = 1 \qquad \forall d \in D, \ \forall f \in F \tag{21}$$

$$x_{dpf} \ge \theta_{pf} x_{dp0} \qquad \forall d \in D, \forall p \in P, \forall f \in F \tag{22}$$

$$y_{svf} \ge \theta_{pf} y_{sv0} \qquad \forall s \in S, \forall v \in V, \forall f \in F \tag{23}$$

Finding the optimal solution of the service migration problem is highly complex mostly due to the introduction of multiple failure scenarios, and it can not be easily found in a reasonable time even for small network instances. Therefore, we apply the column generation method in Migration-LP, adding the candidate paths iteratively to reduce the initial number of variables and constraints to be considered. To be able to apply the method, we use the linear relaxation of binary variables and thus solve the problem as an LP (rather than an ILP).

After solving an initial instance of the service migration problem with a limited set of paths, including working and backup paths, which are found beforehand, we add a new set of candidate paths that possibly improve the objective value. This process is shown as Step 3 and 4 in Fig. 2. To select a candidate path p^*, we define the reduced cost function (24) derived from the Langrangian function (25) of the model using the dual variables of LP[2]. As the paths with positive reduced cost can contribute to the existing feasible solution taken from Step 3, those are added to the used set of paths. Then, the LP is re-solved with the extended set of paths.

$$c_{p^* f} = -\left(\sum_{d \in D} \sum_{e \in E} \alpha_{p^* e} \nu_{ef} \theta_{p^* f} h_d - |p^*| \right) \tag{24}$$

[2]Further details for the column generated method, dual variables, and cost functions can be found in many studies such as [41]–[43].

$$\mathcal{L}^*(x_{dpf}, \eta_{dpf}^{uv}, \nu_{ef}, \rho_{dpf}, \pi_{dpf}, \phi_e) =$$
$$-\sum_{p\in P}\sum_{d\in D}\sum_{f\in F} x_{dpf}|p| +$$
$$\sum_{u,v\in V}\sum_{d\in D}\sum_{p\in P}\sum_{f\in F} \eta_{dpf}^{uv}(x_{dpf} - y_{svf}y_{tuf}\theta_{pf}) +$$
$$\sum_{e\in E}\sum_{f\in F} \nu_{ef}(\sum_{d\in D}\sum_{p\in P} x_{dpf}\theta_{pf}\alpha_{pe}h_d - c_e) +$$
$$\sum_{d\in D}\sum_{p\in P}\sum_{f\in F} \rho_{dpf}(\sum_{e\in E} x_{dpf}\alpha_{pe}l_e^* - l_d) -$$
$$\sum_{d\in D}\sum_{p\in P}\sum_{f\in F} \pi_{dpf}(x_{dpf} - \theta_{pf}x_{dp0}) \qquad (25)$$

Lastly, as Step 5 in Fig. 2, if there is no path left to include, e.g., all remaining paths have a non-positive reduced cost, the reduced problem is solved as an ILP (i.e., without linear relaxation) using only the obtained set of all useful paths.

In terms of complexity, Migration-LP has the highest number of constraints and variables in comparison to Bootstrapping-ILP and Backup-ILP. Solving a very similar problem with Bootstrapping-ILP for multiple failure scenarios, it is bounded by $\mathcal{O}(|D||P||F||S|^2|V|^2)$ constraints and $\mathcal{O}(|F|(|D||P| + |S||V|))$ variables.

IV. Heuristic

As described in Section III, three phases constitute the essence of the problem: (i) Service deployment satisfying demands, (ii) shared link protection scheme minimizing the use of link resources, and (iii) service migration scheme for node protection. We propose a 5-step heuristic covering those three phases referring to our optimization scheme. Fig. 4 shows the corresponding steps of the heuristic addressing the same objective as the optimization models. Step 1 (Service deployment) and Step 3 (Finding working paths) correspond to the bootstrapping phase, where the initial service deployment and assignments of working paths are performed, aiming at minimizing the working path lengths. In Step 2 (Shared link protection), backup paths are found for each demand in a way to maximize the shared use of links similar to the shared protection phase. Note that the order of steps for finding working and backup paths are different than the optimization scheme, which is justified in more detail later on in the paper. Step 4 (Assigning remaining demands) plays a complementary role for Steps 1-3 to ensure that the working and backup paths are assigned for all demands with minimal path lengths and maximal backup capacity sharing. Lastly, Step 5 (Finding backup nodes) is to find alternative service deployment schemes utilizing the shortest available paths to set up in the case of failures, corresponding to the service migration phase. In the rest of this section, we describe each step in more detail.

Step 1. Service deployment: In this step, the services are assigned to physical nodes with sufficient resources. As each

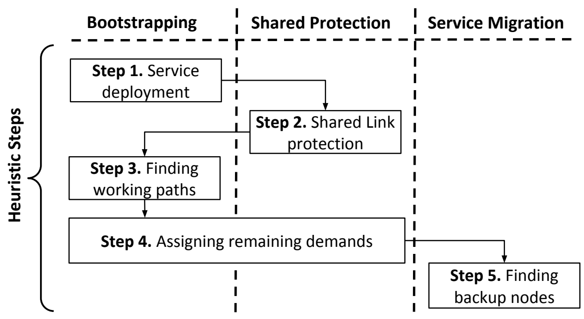

Bootstrapping
Shared Protection
Service Migration
Heuristic Steps
Step 1. Service deployment
Step 2. Shared Link protection
Step 3. Finding working paths
Step 4. Assigning remaining demands
Step 5. Finding backup nodes

Fig. 4: The correspondence between the steps of the heuristic and the optimization phases.

demand is defined between a pair of services, the locations of host nodes are restrained by the latency and data requirements of demands. The host nodes are selected starting from the ones with the highest connectivity, e.g., the highest number of direct neighbors, to ease finding disjoint working and backup paths afterward. For each pair of services utilized by a demand, we ensure that there is at least a path with sufficient link resources and latency cost for the respective demand. Among the alternative deployments, we select the closest nodes for a better quality of service in terms of latency and less link resource consumption at the end.

Note that a service can be utilized by multiple demands. In this case, only a single instance of the related service is placed to the selected node. Accordingly, this node should satisfy latency and data requirements of any demand utilizing that service.

Step 2. Shared link protection scheme: When some of the services are utilized by multiple demands as mentioned in the previous step, they form *chain of services* as it is also seen in Fig. 1. For instance, while a node hosting a service receives data for a traffic demand, that node can send data from the same service to another node to satisfy a different demand. When such services are allocated at physical nodes, it is convenient to define a communication backbone connecting and covering all those nodes to be shared by multiple demands in case of failures. In this step, we utilize our Secondary Backup Backbone (SBB) algorithm to define the chain of services whose host nodes can also be connected sequentially to form the backup backbone. That is, the service chain in the service overlay is also reflected as a chain of physical nodes forming a single backbone, i.e., a connected subgraph. In contrast to the order of optimization phases in Section III, instead of finding working paths first together with the service deployment, we apply a shared link protection scheme before assigning working paths. The reason is that the assignment of working paths restricts the available links to be used in backup paths significantly as they should be disjoint. However, when we maximize the shared use of backup links, i.e., by decreasing the number of used links and the total capacity, there are still sufficient resources left to establish shorter working paths.

Algorithm 1: Secondary Backup Backbone (SBB)

```
1  v_terminal ← $[v_1, v_2, v_3 ... v_n]$
2  v_selected ← $\varnothing$
3  v_Steiner ← $\varnothing$
4  $v$ ← random node $\in V_{\text{terminal}}$
5  v_selected ← $V_{\text{selected}} \cup \{v\}$
6  v_terminal ← $V_{\text{terminal}} \setminus \{v\}$
7  while $v_{terminal} \neq \varnothing$ do
8      $p^* \leftarrow \infty$
9      for $v_1 \in V_{terminal}$ do
10         $\mathrm{v}_2 \leftarrow v_n \in V_{\text{selected}}$ closest to $v_1$
11         if $\mathrm{v}_2$ is close enough to $\forall v_n \in V_{terminal}$ then
12             $p \leftarrow$ secondary shortest path between $v_1$-$v_2$
13             if $p$ is shorter than $p*$ then
14                 $p^* \leftarrow p$
15                 $v \leftarrow v_2$
16     v_selected ← $V_{\text{selected}} \cup \{v\}$
17     v_terminal ← $V_{\text{terminal}} \setminus \{v\}$
18     v_Steiner ← $V_{\text{Steiner}} \cup \{v_n \in p^*\}$
```

For each service chain, SBB constructs a modified Steiner tree [44] on the physical network where each host node is considered as a terminal node and any intermediate node, which belongs to the tree, is a Steiner node. A Steiner tree is defined as a connected subgraph, e.g., a tree, including a set of given nodes, i.e., terminal nodes. All other nodes to be used to connect terminal nodes are called Steiner nodes. Generally, the construction of a Steiner tree refers to finding a minimal subgraph, or the shortest tree, having all terminal nodes with a minimum number of Steiner nodes, and it is known to be an NP-complete problem [45]. However, according to our initial experiments, finding the shortest tree eliminates the possible use of shortest paths as working paths afterward. Since the working paths are the most used ones until a failure occurs, keeping them shorter leads to a better QoS. Therefore, we use a simple heuristic to construct a *secondary* Steiner tree, which utilizes the shortest disjoint path to the shortest path between two nodes, i.e., the secondary shortest path, instead of directly using the shortest one. Algorithm 1 briefly describes those steps of SBB.

Iterating through the elements of $\mathrm{v}_{\text{terminal}}$ including all host nodes obtained from the previous step, SBB forms the secondary shortest paths between the nodes verifying that the latency requirement for each demand is satisfied for the backup communication through the backbone. In line 11, SBB ensures that v_2 does not violate such requirements for a demand between the services on v_2 and any other terminal nodes in $\mathrm{v}_{\text{selected}}$. Then, the nodes in the shortest path excluding the end hosts, e.g., terminal nodes, are added to the set $\mathrm{v}_{\text{Steiner}}$ as Steiner nodes. After connecting each node in $V_{\text{selected}} \cup V_{\text{Steiner}}$, SBB returns that secondary Steiner tree satisfying demand requirements.

Step 3. Finding working paths: After forming the backup backbone in Step 2, we use the Mutually Disjoint Paths (MDP) algorithm to calculate working paths in this step. Algorithm 2 shows how MDP calculates the working paths for demands whose backup paths are defined in a particular backup backbone G_{Steiner}.

Algorithm 2: Mutually Disjoint Paths (MDP)

```
1  for $d \in D$ do
2      $\mathrm{v}_1, \mathrm{v}_2 \leftarrow$ Host nodes of $d$
3      $p_d^w \leftarrow$ Backup segment between $\mathrm{v}_1 - \mathrm{v}_2 \in G_{\text{Steiner}}$
4      for $p \in P_d$ do
5          if $p$ and $\{p_q^w \mid \forall q \in H_d\}$ disjoint then
6              if $p$ and $p_d^b$ disjoint then
7                  if $p$ satisfies $l_d$ then
8                      $p_d^w \leftarrow p$
```

There are three essential aspects to be considered when calculating working paths. First, they should be mutually disjoint if the backup paths of the respective demands are shared. Even though we have formed a single backbone, each demand $d \in D$ uses only a segment of the backbone G_{Steiner}, i.e., a path between the nodes hosting the services of that demand, that could be shared or not. Therefore, when finding a working path, we first check if d utilizes any capacity shared with another demand $q \in H_d$ with an assigned working path w_q and ensure that they are disjoint (line 5). The second issue is, the working path of d, p_d^w should be disjoint to the respective segment of the backup backbone p_d^b that is used that demand (line 6). Lastly, p_d^w should satisfy the latency requirements of d (line 7). Note that similar to the optimization scheme, we have found all paths in advance and added them to the problem as an input. Therefore, checking the disjointness of paths is a matter of comparison between their nodes and links.

Fig. 5 shows an example construction of a backup backbone and the respective working paths. Initially, three services are deployed to v_1, v_2, and v_3 where two demands are defined between v_1-v_2 and v_1-v_3. Using additional Steiner nodes v_4 and v_5, SBB constructs the backup backbone among the nodes v_1-v_5. Accordingly, MDP calculates two disjoint working paths v_1-v_6-v_7-v_2 and v_1-v_8-v_3. Note that those paths are both mutually disjoint and concerning their relation with the backbone. In the backbone, the links between v_1-v_4 and v_4-v_5 are shared.

Step 4. Assigning remaining demands: As a result of Step 2 and 3, the working and backup paths are calculated around backbones tightly for each service chain. In particular cases such as the lack of disjoint paths and limited available link resources, some demands may not be assigned according to the initial service deployment in Step 1. To cope with such scenarios, we utilize an improved version of our previous heuristic Random Deployment with Disjoint Paths (RDDP) in [10]. RDDP is a greedy heuristic consisting of two phases,

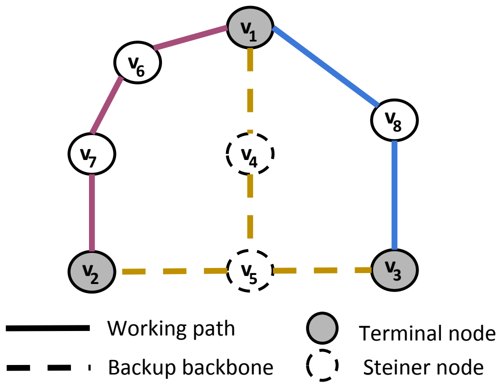


Fig. 5: Backup backbone (dashed black) with host (terminal) nodes v_1, v_2, v_3, Steiner nodes (dashed) v_4, v_5, and respective working paths (straight purple and blue). Two demands are defined between nodes v_1-v_2 and v_1-v_3. In this case, the links between v_1-v_4 and v_4-v_5 are shared between those demands.

service deployment and routing. In the first phase, it allocates the services to a randomly selected pair of nodes not hosting other services. If every node hosts at least one service, RDDP selects the node with the highest available resources. We improve RDDP to select the nodes whose secondary shortest path maximizes the use of shared capacity. In the second phase, it allocates two node-disjoint paths for the inter-service demand between selected nodes, one for the main use and the other one as a redundant backup. If disjoint paths cannot be found between those nodes, they are reselected by following the same greedy approach.

Note that RDDP selects the feasible nodes and paths for each demand after a limited number of trials where the pairs of nodes are examined randomly as it is explained in [10]. The design parameter $limit$ can be selected according to the network size as the number of possible 2-combinations of nodes is proportional. In our experiments, we applied $limit = 100$.

Step 5. Finding backup nodes: Although we have established backup paths for protection against single link failures so far, any failure occurring in host nodes can still disrupt the communication as the hosted services would fail. In the last step of the heuristic, we utilize another heuristic from our former work, namely Backups with Secondary Redundant Path (BSRP). It simply finds an alternative node for each host node in the initial deployment to migrate its services in the case of a failure.

When an alternative node is selected to host a service, three criteria are important to satisfy the requirements of all demands which utilize that service: The alternative node should (i) have sufficient resource capacity to host the respective service, (ii) have required paths with sufficient capacity in-between the nodes hosting the other services for related demands, and (iii) be in a position to comply with the latency requirements of all related demands. In BSRP, we search for alternative nodes starting from the ones with the highest

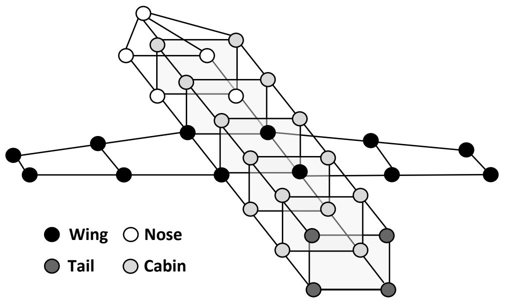


Fig. 6: A potential in-plane topology.

remaining resource capacity for each service. Among the candidates, the node with the minimum total length of paths to the other services for the respective demands utilizing the migrated service is selected. Eventually, the alternative paths to be used after a service migration consume network resources minimally.

V. Evaluation

To measure the performance of our optimization scheme and heuristic, we considered a number of scenarios and used a set of metrics. In this section, we present our experimental setup and discuss our numerical results in detail.

A. Experiment Setup

In this section, we describe (i) our computational resources and tools used to run our experiments, (ii) our topology and service overlay generation approach, (iii) what we measured and related parameters, and lastly (iv) the metrics we used for the comparisons.

1) Computational Resources: The optimization models were implemented in CPLEX 12.7.0, and all experiments were conducted in a server with 64-core Intel Xeon 2.10GHz CPU and 256GB RAM. The resource utilization varied for the different phases of the optimization scheme. For the largest instances of the problem, i.e., 35 nodes and 21 demands, phase 3 (service migration) kept the CPU utilization around the level of 80% for all the cores and used all available RAM. As we used pre-computed paths in the model, it also occupied an amount of memory proportional to the network size and connectivity. On the contrary, phase 1 and 2 were executed more easily using the CPLEX branch-and-bound method and utilizing the presolver.

2) Topology and Overlay Generation: We evaluated the performance of the optimization models and the heuristic for different types of topologies and service overlays. First, we created a potential in-plane topology shown in Fig. 6 as avionics is one of the key safety-critical domains to make use of service-based flexible network design. In this topology, we considered a cabin network that is interconnected with the nose, tail, and wings of the plane. Note that in traditional in-plane networks, there might be tens of end-systems, subsystems, and hundreds of signals between critical components [46]. Here, we included the cabin also giving services with

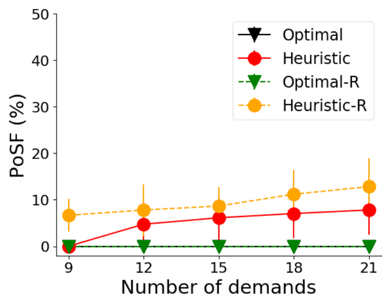


Fig. 7: Probability of service failure in case of a single node failure.

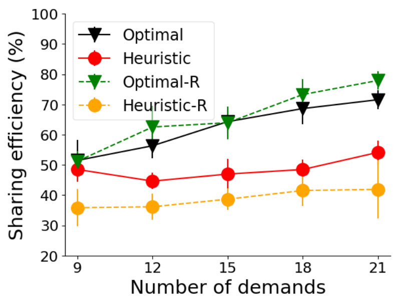


Fig. 8: Sharing efficiency.

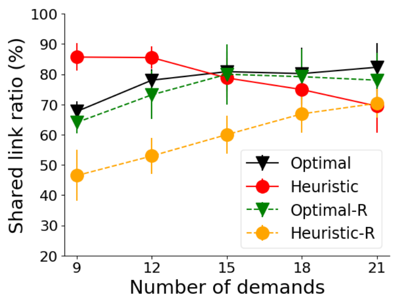


Fig. 9: Shared link ratio.

higher traffic loads, e.g., infotainment, and combined it with the rest of the network to have a complete model of a connected aircraft. The entire topology has 35 nodes with different service-hosting capabilities and the average node degree of 3.7. Eventually, this network can be extended to any other mission-critical domain having varying traffic demands to be satisfied within a bounded latency.

For the given in-plane topology, we defined a service overlay with certain characteristics. Tables III and IV show those characteristics for the services and demands, respectively. In Table III, we present three types of services concerning their resource demand, criticality, possible position to be placed in the topology, and their quantity. The set S_1 consists of low resource demand and high criticality, e.g., control signals, that can be in nose, tail, and cabin. A quarter of all the services belongs to that type. S_2 represents medium resource demands and criticality services that can be placed anywhere. Lastly, S_3 consists of the services with high-resource demand and low-criticality to be placed in cabin and half of the services are defined in that type. We generated the exact values, e.g., traffic and latency requirements of demand, for each type of service randomly keeping their interrelation.

TABLE III: Characteristics of the services that are used in the experiments.

Service Set	Resource demand	Criticality	Position	Quantity
S_1	Low	High	Nose, cabin, tail	$\|S\|/4$
S_2	Medium	Medium	Any	$\|S\|/4$
S_3	High	Low	Cabin, tail	$\|S\|/2$

We considered three types of demands shown in Table IV utilizing the types of services from Table III. D_1 consists of demands with low data traffic and being time-sensitive, e.g., with tight latency constraints, and defined between the services of S_1, i.e., the most critical ones. D_2 represents the demands with medium data traffic with best-effort QoS between the services of S_1, S_2, and S_3. Lastly, D_3 represents the high-traffic demands, mostly defined for the cabin part. We defined half of the demands in D_3 as proportional to the number of nodes in the cabin part, and the rest of the demands were equally distributed between D_1 and D_2.

Apart from a very regular in-plane topology where the majority of the nodes have similar connectivity, we also used random networks with the same number of nodes and similar connectivity. As nodes do not have particular roles, e.g., either in nose, cabin, tail, or wings, we did not restrict the position of the services. That is, we used the same characteristics for the service overlay, excluding the positional constraints.

TABLE IV: Characteristics of the demands that are used in the experiments.

Demand Set	Traffic demand	Latency	Services	Quantity
D_1	Low	Time-sensitive	$S_1 \leftrightarrow S_1$	$\|D\|/4$
D_2	Medium	Best-effort	$S_1 \leftrightarrow S_2$, $S_2 \leftrightarrow S_2$, $S_3 \rightarrow S_2$	$\|D\|/4$
D_3	High	Best-effort	$S_3 \leftrightarrow S_3$	$\|D\|/2$

3) Measurements and Parameters: For most of the experiments, we measured *Optimal* and *Heuristic* values that represent the optimal and the heuristic's results for the given in-plane topology, *Optimal-R* and *Heuristic-R*, in contrast, show the results for random networks of the same size.

We evaluated the models and the heuristic for the increasing number of demands. In the end, we also show the scalability of the heuristic for the increasing number of nodes generating larger random topologies (50-70 nodes) with the average node degree of 2.8. In the scalability scenarios, we used a fixed-size service overlay with 50 demands. For each scenario, the experiments were repeated 30 times and the results are given with a 95% confidence interval. For all experiments, the optimality gap was defined as 5%, which means the results could deviate from the optimum at most by 5%.

4) Metrics: We used the metrics listed below to evaluate the performance of all phases of the optimization scheme and the heuristic.

Probability of service failure: It is the ratio of the number of services that cannot be migrated to an alternative node to all services.

Sharing efficiency: It is the ratio of the difference between shared and dedicated backup capacity to the dedicated backup capacity without sharing. The latter is calculated by disabling capacity-sharing and reserving dedicated capacity on the configured backup paths. The sharing efficiency represents the capacity gain by sharing. We compare our capacity sharing ap-

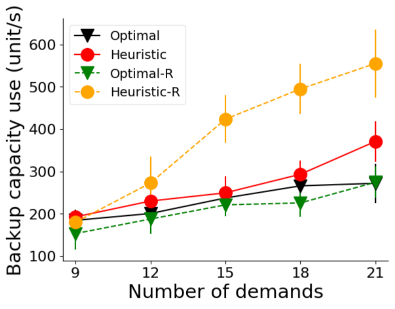


Fig. 10: Total backup capacity use.

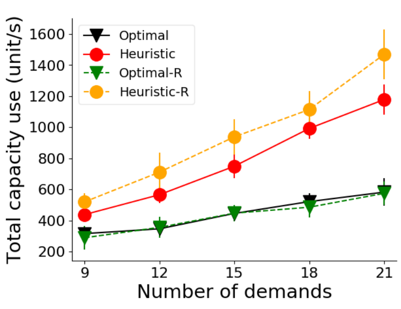


Fig. 11: Total capacity use.

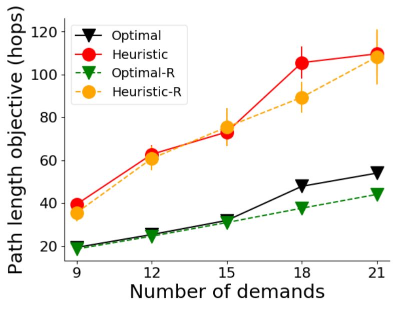


Fig. 12: Total length of bootstrapping and service migration phases.

proach with the dedicated one as it is a broadly-used approach for the redundancy in mission critical systems. Besides, this comparison helps to emphasize the improvements made by us following our preliminary work, where we used the dedicated backup capacity approach.

Shared link ratio: It is the ratio of the shared links to all backup links. The sharing ratio measures the efficiency of backup capacity allocation where the backup links can be shared only if the respective working paths are mutually disjoint. Any link utilized as a backup link by multiple demands is counted as a shared link.

Total length of working paths: As Bootstrapping-ILP and Migration-LP optimize the total path length in terms of number of hops, this metric represents the total objective value of those two phases.

Backup capacity use: It is the total link capacity, i.e., bandwidth, required for all the backup paths. As Backup-ILP minimizes the use of backup capacity by utilizing shared capacity, this metric also represents the resulting value of the objective function.

Total capacity use: It is the total link capacity required for all the paths, i.e., both working and backup paths. Especially for the heuristic, it shows the efficiency of working path selection.

Prolongation factor: Prolongation factor between two paths is the ratio of the length of one path to the other's length. We consider (i) backup to working path and (ii) selected backup to the ideal backup path prolongation factors to show the balance between backup and working paths, and the efficiency of the selected paths, respectively.

Solution time: It is measured for each individual phase of the optimization scheme. As the heuristic solves the target problem in a neglectable time, e.g., seconds to a few minutes, the solution time is considered for only the optimization scheme to evaluate the increasing complexity by the number of demands.

B. Results

In this section, we present the experiment results using the selected metrics and scenarios for different topologies and service overlays.

1) Fault-tolerance: Fig. 7 shows the probability of service failure (PoSF) during a single node failure as a function of the number of demands. As the optimal solution guarantees to find a backup path for each demand against link failures and a backup node for each service against node failures, it protects the network against all single link failures and any single failure of the host nodes. Therefore, the optimal deployments at both in-plane and random topology result in 0% PoSF. Similarly, in our experiments, our heuristic is also successful to reserve the required capacity in backup paths for all the demands. However, in the last stage of the heuristic, there are some scenarios where it fails to find alternative nodes to migrate services due to (i) insufficient amount of node resource capacity, (ii) link capacity, or (iii) lack of suitable paths satisfying the latency requirements. For the in-plane topology and random networks, our heuristic keeps the PoSF below 5% and 10%, respectively. As the service deployment may spread through the network without node capability constraints and random connectivity, Step 5 in the heuristic fails more often to migrate all services in random networks satisfying especially case (iii) above.

Note that some mission-critical services cannot tolerate any failure at all and require replicated hardware or software to ensure seamless failover in case of single failures. Our heuristic can be considered as an additional fault-tolerance mechanism for such cases, thus avoiding replication costs. Furthermore, our optimization model can be used to compute the desired level of fault-tolerance for all services. Hence, it can be used during the network design stage to plan the whole backup scheme despite its longer solution time.

2) Sharing Efficiency: Fig. 8 shows the sharing efficiency, which is the gain of using the shared capacity instead of dedicated capacity, depending on an increasing number of demands. While the optimization scheme (only Backup-ILP) can utilize backup paths to decrease the required backup capacity by 50-75%, our heuristic gives steady results around 50% and 40% backup capacity savings for in-plane (*Heuristic*) and random (*Heuristic-R*) topologies.

Similar to sharing efficiency, Fig. 9 shows the shared link ratio to measure the effective use of backup paths for an increased shared capacity. The optimal deployment results in 60-80% of the backup links used by several backup paths at once. For the in-plane topology, our heuristic results in 70-85% shared link use, which is quite similar to the results of the optimization scheme. Note that since maximizing the shared link ratio is not the objective of the optimization scheme, the heuristic can give better results for small number of demands. However, with an increasing number of demands, while the result of the optimization scheme converges to 80% shared link ratio, our heuristic shows a decreasing trend as it gets harder to find Steiner trees promoting mutually disjoint working and backup paths. On the other hand, the establishment of disjoint working paths is easier even for an increasing number of demands in random networks as node capability constraints are neglected, and demands can be placed more flexibly. In that case, *Heuristic-R* results in 45-70% shared link ratio.

Considering Fig. 8 and Fig. 9, the number of shared links does not have to be proportional to the sharing efficiency as the latter also depends on the load and sharing links for small loads may not affect the efficiency so much.

3) Objective Functions: As presented in Section III, there are two different objectives for the three phases of our optimization scheme: (i) Minimizing the path lengths that are used as working paths at bootstrapping and after service migration, and (ii) minimizing the reserved resources for backup capacity. Figs. 12 and 10 show the comparison of the optimal solution and the results of our heuristic in terms of those two objectives.

Fig. 10 shows the total backup capacity utilization referring to the objective function (8). For the in-plane topology, our *Heuristic* give near-optimal results. However, for random topology, when we removed node capability restriction, e.g., any service can be placed to any node, the heuristic (*Heuristic-R*) may tend to spread the services to the farther nodes, which may result in longer Steiner trees. It is also related to the $limit$ parameters as we examine a limited number of node combinations among the suitable nodes, e.g., having the capability to host a certain service, and select the best ones. This variety in possible changes can also be shown in the larger confidence interval in *Heuristic-R*. Such difference is also reflected in Fig. 11 and our heuristic requires a slightly higher capacity in random networks than in the more structured in-plane topology.

Fig. 12 shows the length of all working paths representing the results of the objective functions (1) and (15) for an increasing number of demands. For both in-plane and random topologies, the optimal results and the one from our heuristic scale proportionally. Even though we build secondary Steiner trees via SBB as described in Section IV instead of the shortest ones, some of the possible shortest paths are still used by the backup paths in the heuristic before constructing the working paths. Moreover, longer working paths might be utilized. However, as we embed latency requirements for demands to both working and backup finding processes, in our heuristic, the upper bound for the length of each working path is always restricted to satisfy such requirements.

4) Path Selection: The differences between the length of working and backup paths indicates the possible degradation in the QoS in case of a failure. That is, when a backup path is longer than the respective working path, shifting from a working to a backup path might result in an increased

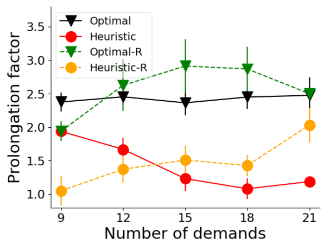


Fig. 13: Prolongation w.r.t working paths.

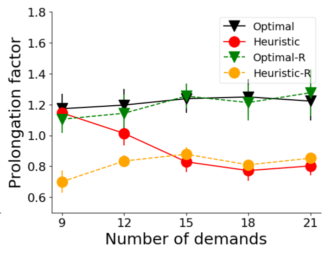


Fig. 14: Prolongation w.r.t. ideal backup paths.

communication delay. Selecting shorter backup paths may consume resources that could be normally utilized by working paths. Fig. 13 shows the prolongation factor for backup and working paths (i.e., the ratio of the length of backup paths to working paths). The optimal solution on the in-plane topology (itOptimal), results in backup paths that are almost constantly 2.5 times longer than the corresponding working paths. In contrast, in random networks, the prolongation factor stays within 2 and 3 with larger confidence intervals. The structure of the random network seems to have a significant influence on the results. For the heuristic, as we first obtain backup paths by finding a backup backbone, some of the shorter paths are used for the backup before they can be chosen as working paths. It results in a greater number of working and backup paths of similar length, which manifests in prolongation factors in between one and two.

Fig. 14 shows the prolongation factor between the obtained backup paths and the *ideal* backup paths. An ideal backup path is defined as the secondary shortest path that is disjoint to the shortest path between two nodes. In the best case, where only a few demands exist in a large network, such a configuration for working and backup paths would give the best QoS and protect against any single link failure. As seen in Fig. 14, the optimal configuration provides 1.2-1.3 times longer backup paths in comparison to the ideal one. Our heuristic usually calculates shorter backup paths as it utilizes the backup backbone first. Therefore, the prolongation factor is 0.7-1.2, occasionally below 1.0.

5) Scalability: We evaluated the scalability of our heuristic for some of the selected metrics for an increasing topology size. Fig. 15 shows the sharing efficiency for 50 demands depending on the node count. Apart from calculating the total backup load on the selected segments of the backup backbone considering shared and dedicated backup capacity (*Selected backup path*), according to our definition of sharing efficiency, we also considered another scenario where we reserved dedicated backup capacity on the ideal backup path (*Ideal backup path*) as it is defined in Section V-B4. The figure shows that our heuristic can again provide 50% capacity gain even for larger networks similar to the results for small topologies. Moreover, our shared backup capacity scheme seems to achieve 40-50% gain compared to a dedicated backup scheme for the ideal paths.

Fig. 16 shows that the shared link ratio increases up to 70%

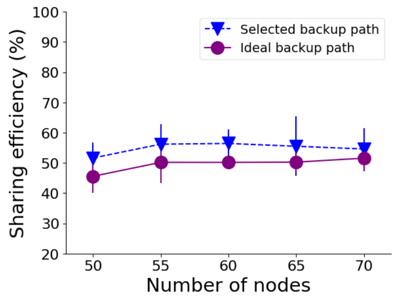


Fig. 15: Sharing efficiency with respect to ideal and selected backup paths.

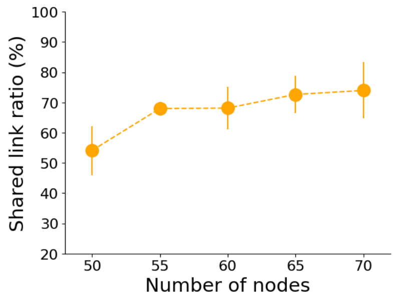


Fig. 16: Shared link ratio.

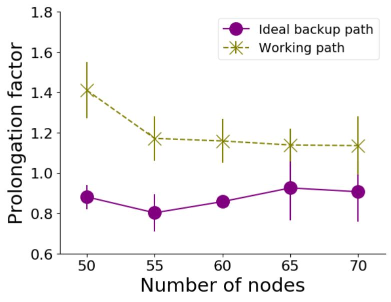


Fig. 17: Prolongation with respect to ideal backup paths and working paths.

with increasing topology size implying larger set of potential backup paths. Lastly, we show the prolongation factor of shared backup paths with respect to the ideal backup paths and working paths combining in Fig. 17. As can be seen in the figure, for larger networks, our heuristic can find backup paths close to the ideal ones, i.e., with a prolongation factor of nearly 1.0, and better working paths with a lower prolongation factor 1.6 to 1.2 in comparison to smaller networks where this factor goes up to 2.0.

6) Solution Time: The phases of the optimization scheme require a considerable amount of time when increasing the size of the service overlay. As we have summarized in the complexity discussion of our previous work [10], finding the optimal configuration, which is resilient to all single node failures without considering any shared capacity, might take days as it is formulated as a single linear problem. Here, with three improvements, namely (i) dividing the whole optimization problem into three phases, (ii) finding disjoint backup paths against link failures instead of finding a new configuration for each failure and (iii) applying column generation method for the reduced problem (i.e., a fewer number of failure scenarios addressing only the failure of the host nodes), we improved the solution time considerably. Table V shows the problem size in terms of number variables and constraints and the solution time per each optimization phase.

Eventually, the service migration phase has more constraints and variables (when all the paths are added at the end of the column generation) than the other phases as it finds multiple configurations for different single node failure scenarios. Therefore, it was the decisive phase for the overall solution time with durations from 18 minutes to 11.5 hours. The overall solution took from 21 minutes to 14 hours, depending on the overlay size. It is a significant improvement for three times larger networks in comparison to our previous work [10].

VI. Conclusion

Mission-critical systems typically include several safety-critical services, which require considering their resilience against failures and attacks and already at the design stage. In this study, we present a service-based network design in which we embed an overlay of services and the traffic demands as well as QoS requirements between them in a mission-critical embedded network in a fault-tolerant manner. Extending our previous work in [10], we formulate the joint service deployment and routing problem as a series of optimization models to obtain a resource-efficient and resilient network model that provides fault-tolerance against single link and node failures leveraging shared backup protection and service migration schemes. To solve larger problem instances, we apply the column generation method in our optimization models. As finding the optimal solution for that joint problem is known to be NP-hard, we also propose a heuristic. We have evaluated the performance of the optimization scheme and the heuristic in different scenarios with various metrics. Our experiment results indicate that our heuristic can allocate near-optimal backup capacity offering up to 50% capacity gain compared to using dedicated backup capacity. Moreover, while our optimization scheme can find service configurations that are completely fault-tolerant to single link and node failures, our heuristic can tolerate the single failure of 90% of all nodes.

In the future work, we plan to extend our objectives focusing on the different aspects of the network design, e.g., minimum energy consumption and minimum node or link deployment, as well as examining the impact of relevant parameters, e.g., the optimality gap.Moreover, although we assumed a single-failure model in this study as it is frequently considered in the literature, we intend to address more advanced models, e.g., correlated and cascading failures, under different assumptions such as shared risk link and node groups.

References

[1] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, “Intelligent Manufacturing in the Context of Industry 4.0: A Review,” *Engineering*, vol. 3, no. 5, pp. 616 – 630, 2017.

[2] A. Koenig, “Integrated Sensor Electronics with Self-X Capabilities for Advanced Sensory Systems as a Baseline for Industry 4.0,” in *19th ITG/GMA-Symposium of Sensors and Measuring Systems*, pp. 1–4, 2018.

[3] O. Burkacky, J. Deichmann, G. Doll, and C. Knochenhauer, *Rethinking Car Software and Electronics Architecture*. McKinsey & Co, 2020.

[4] *Automotive Virtual Platform Specification*. GENIVI, Jul 2020.

[5] *Future Airborne Capability Environment (FACE) Technical Standard Edition 3.1*. The Open Group, Jul 2020.

[6] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, “Wireless Sensor Network Virtualization: A Survey,” *IEEE Comm. Surveys and Tutorials*, vol. 18, no. 1, pp. 553 – 576, 2016.

[7] M. Nkomo, G. P. Hancke, A. M. Abu-Mahfouz, S. Sinha, and A. J. Onumanyi, “Overlay Virtualized Wireless Sensor Networks for Application in Industrial Internet of Things: A review,” *Sensors*, vol. 18, no. 10, pp. 1–33, 2018.

TABLE V: Problem size and solution time values for the given topology from Fig. 6 and the increasing number of demands

# demands	Bootstrapping-ILP			Backup-ILP			Migration-LP		
	Variables	Constraints	Sol. Time	Variables	Constraints	Sol. Time	Variables	Constraints	Sol. Time
9	262314	344840	2.5m	223614	9593	41s	786744	1377508	18.4m
12	362184	515016	9.7m	298152	18474	54s	1676340	2995976	74.6m
15	483270	712880	16m	372690	19467	65s	3257940	6166825	3h
18	607572	938372	28.3m	447228	31989	103s	3663252	7277190	9h
21	741090	1191512	88.3m	521766	37935	204s	5736780	11920629	11.4h

[8] K. Ogawa, H. Sekine, K. Kanai, K. Nakamura, H. Kanemitsu, J. Katto, and H. Nakazato, “Performance Evaluations of IoT Device Virtualization for Efficient Resource Utilization,” in *Global IoT Summit (GIoTS)*, pp. 1–6, 2019.

[9] P. Karhula, J. Janak, and H. Schulzrinne, “Checkpointing and Migration of IoT Edge Functions,” in *2nd Int. Workshop on Edge Systems, Analytics and Networking*, EdgeSys, pp. 60–65, ACM, 2019.

[10] D. Ergenc, J. Rak, and M. Fischer, “Service-Based Resilience for Embedded IoT Networks,” in *50th IEEE/IFIP International Conf. on Dependable Systems and Networks (DSN)*, pp. 540–551, 2020.

[11] G. Heiser, “Virtualizing embedded systems – why bother?,” in *48th ACM/EDAC/IEEE Design Automation Conf. (DAC)*, pp. 901–905, 2011.

[12] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth, “Modeling and Placement of Cloud Services with Internal Structure,” *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, pp. 429–439, 2016.

[13] D. Breitgand, A. Marashini, and J. Tordsson, “Policy-driven service placement optimization in federated clouds,” *IBM Research Division, Tech. Rep*, vol. 9, pp. 11–15, 2011.

[14] L. Pu, L. Jiao, X. Chen, L. Wang, Q. Xie, and J. Xu, “Online Resource Allocation, Content Placement and Request Routing for Cost-efficient Edge-caching in Cloud Radio Access Networks,” in *IEEE Journal on Selected Areas in Communications*, vol. 36, pp. 1751–1767, 2018.

[15] M. B. Gawali and S. K. Shinde, “Task Scheduling and Resource Allocation in Cloud Computing using a Heuristic Approach,” *Journal of Cloud Computing*, vol. 7, no. 1, 2018.

[16] B. Addis, D. Belabed, M. Bouet, and S. Secci, “Virtual Network Functions Placement and Routing Optimization,” in *IEEE Int. Conf. on Cloud Networking, CloudNet*, pp. 171–177, IEEE, 2015.

[17] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, “Orchestrating Virtualized Network Functions,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, 2016.

[18] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, “On Dynamic Service Function Chain Deployment and Readjustment,” *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 543–553, 2017.

[19] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, “Piecing Together the NFV Provisioning Puzzle: Efficient Placement and Chaining of Virtual Network Functions,” in *IFIP/IEEE Int. Symp. Integrated Netw. Mgmt. (IM)*, pp. 98–106, 2015.

[20] G. Lee, M. Kim, S. Choo, S. Pack, and Y. Kim, “Optimal Flow Distribution in Service Function Chaining,” in *ACM International Conference Proceeding Series*, vol. 08-10-June-2015, pp. 17–20, 2015.

[21] M. Médard, S. G. Finn, R. A. Barry, and R. G. Gallager, “Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, pp. 641–652, 1999.

[22] P. P. Lee, V. Misra, and D. Rubenstein, “Distributed algorithms for secure multipath routing in attack-resistant networks,” *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, pp. 1490–1501, 2007.

[23] I. B. Barla, D. A. Schupke, M. Hoffmann, and G. Carle, “Optimal Design of Virtual Networks for Resilient Cloud Services,” in *Int. Conf. on Design of Reliable Comm. Networks (DRCN)*, pp. 218–225, 2013.

[24] J. Xu, J. Tang, K. Kwiat, W. Zhang, and G. Xue, “Survivable Virtual Infrastructure Mapping in Virtualized Data Centers,” in *IEEE 5th International Conference on Cloud Computing*, pp. 196–203, June 2012.

[25] M. T. Beck, J. F. Botero, and K. Samelin, “Resilient Allocation of Service Function Chains,” in *IEEE Conf. on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 128–133, 2016.

[26] “Fault-Resilient Topology Planning and Traffic Configuration for IEEE 802.1Qbv TSN Networks,” in *IEEE 24th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 151–156, 2018.

[27] Xiaolong Xie, Xin Li, Shanguo Huang, Bingli Guo, Shan Yin, Qian Kong, Tao Gao, and Wensheng Zhai, “Design for Shared Backup Path Protection based on Content Connectivity Against Disaster in Elastic Optical Datacenter Networks,” in *15th International Conference on Optical Communications and Networks (ICOCN)*, pp. 1–3, 2016.

[28] J. Rak, “Fast Service Recovery Under Shared Protection in WDM Networks,” *Journal of Lightwave Technology*, vol. 30, no. 1, pp. 84–95, 2012.

[29] B. Todd and J. Doucette, “Demand-wise Shared Protection Network Design and Topology Allocation with Dual-failure Restorability,” in *11th International Conference on the Design of Reliable Communication Networks (DRCN)*, pp. 73–80, 2015.

[30] F. He, T. Sato, and E. Oki, “Survivable Virtual Network Embedding Model with Shared Protection over Elastic Optical Network,” in *IEEE Int. Conf. on Cloud Networking (CloudNet)*, pp. 1–3, 2019.

[31] J. Chenni Kumaran and M. Aramudhan, “A Survey on Resource Allocation Strategies in Cloud,” *International Journal of Reasoning-based Intelligent Systems*, vol. 10, no. 3-4, pp. 328–336, 2018.

[32] N. K. Pandey, S. Chaudhary, and N. K. Joshi, “Resource Allocation Strategies used in Cloud Computing: A Critical Analysis,” in *IEEE Conf. on Communication, Ctrl. and Intelligent Syst. (CCIS)*, 2017.

[33] X. Li and C. Qian, “A Survey of Network Function Placement,” pp. 948–953, 2016.

[34] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, “A Comprehensive Survey of Network Function Virtualization,” *Computer Networks*, vol. 133, pp. 212–262, 2018.

[35] Canhui Ou, Jing Zhang, Hui Zang, L. H. Sahasrabuddhe, and B. Mukherjee, “New and improved approaches for shared-path protection in WDM mesh networks,” *Journal of Lightwave Technology*, vol. 22, no. 5, pp. 1223–1232, 2004.

[36] G. Ellinas, D. Papadimitriou, J. Rak, D. Staessens, J. P. Sterbenz, and K. Walkowiak, “Practical issues for the implementation of survivability and recovery techniques in optical networks,” *Optical Switching and Networking*, vol. 14, pp. 179 – 193, 2014. Special Issue on RNDM'13.

[37] G. P. Mccormick, “Computability of Global Solutions to Factorable Nonconvex Programs: Part I – Convex Underestimating Problems,” *Math. Program.*, vol. 10, p. 147–175, Dec. 1976.

[38] S. Ramamurthy, L. Sahasrabuddhe, and B. Mukherjee, “Survivable WDM Mesh Networks,” *Journal of Lightwave Technology*, vol. 21, no. 4, pp. 870–883, 2003.

[39] J. Rak and D. Hutchison, eds., *Guide to Disaster-Resilient Communication Networks*. Computer Comm. and Networks, Springer, 2020.

[40] J. Tapolcai, P. Ho, D. Verchere, T. Cinkler, and A. Haque, “A New Shared Segment Protection Method for Survivable Networks with Guaranteed Recovery Time,” *IEEE Transactions on Reliability*, vol. 57, no. 2, pp. 272–282, 2008.

[41] M. Ruiz, M. Pióro, M. Żotkiewicz, M. Klinkowski, and L. Velasco, “Column Generation Algorithm for RSA Problems in Flexgrid Optical Networks,” *Photonic Network Communications*, vol. 26, no. 2-3, pp. 53–64, 2013.

[42] C. Rocha and B. Jaumard, “A Unified Framework for Shared Protection Schemes in Optical Mesh Networks,” *Pesquisa Operacional*, 2007.

[43] L. S. Lasdon, “Duality and decomposition in mathematical programming,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 86–100, 1968.

[44] K. Makki and N. Pissinou, “The Steiner Tree Problem with Minimum Number of Vertices in Graphs,” in *2nd Great Lakes Symposium on VLSI*, pp. 204–206, 1992.

[45] A. Biniaz, A. Maheshwari, and M. Smid, “On the hardness of full Steiner tree problems,” *Journal of Discrete Algorithms*, vol. 34, 2015.

[46] B. Annighoefer, C. Reif, and F. Thieleck, “Network Topology Optimization for Distributed Integrated Modular Avionics,” in *IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC)*, 2014.

Appendix C

Distributed Bio-inspired Configuration of Virtualized Mission-critical Networks

Abstract

Modern mission-critical embedded systems such as autonomous cars and avionics consist of a multitude of interconnected nodes and services with various QoS requirements. Virtualization provides further flexibility, configurability, and isolation to such systems by enabling dynamic service placement to off-the-shelf virtualized hardware. Although the service-oriented systems usually rely on a single centralized controller for the service configuration and maintenance as well as establishing their inter-communication, more autonomous and self-driven control schemes are required to cope with their increasing scalability and heterogeneity. Accordingly, bio-inspired algorithms (BIAs) offer distributed self-organization methods by adapting the natural phenomena such as collaborating bee and ant colonies for the requirements of modern networked systems. In this paper, we leverage ant-colony optimization to solve the joint service allocation and routing (JSAR) problem distributedly, where mixed-criticality services should be distributed and communicated under strict QoS requirements on a virtualized, physical network. We also introduce an integer linear program (ILP) to find the optimal solution for JSAR that can be computed by a centralized controller. Our experiments show that our heuristics successfully solve JSAR for even scaling scenarios. Besides, utilizing different redeployment strategies, they can be adapted to obtain near-optimal results in terms of resource efficiency.

Reference

Doğanalp Ergenç, D. Sorejevic, M. Fischer. Distributed Bio-inspired Configuration of Virutalized Mission-critical Networks. IEEE Global Communications Conference (GLOBECOM), 2022.

Contribution

In the given publication, the main contribution belongs to this thesis. The second co-author evaluated an early version of the proposed framework in the context of his master's thesis. However, it is here remodeled from scratch and extended with an optimization model. The evaluation is also re-conducted with new metrics. The third co-author helped to improve the quality of the paper with his valuable feedback.

Distributed Bio-inspired Configuration of Virtualized Mission-critical Networks

Doganalp Ergenc, David Sorejevic, Mathias Fischer
University of Hamburg, Germany
name.surname@uni-hamburg.de

***Abstract*—Modern mission-critical embedded systems such as autonomous cars and avionics consist of a multitude of interconnected nodes and services with various QoS requirements. Virtualization provides further flexibility, configurability, and isolation to such systems by enabling dynamic service placement to off-the-shelf virtualized hardware. Although the service-oriented systems usually rely on a single centralized controller for the service configuration and maintenance as well as establishing their inter-communication, more autonomous and self-driven control schemes are required to cope with their increasing scalability and heterogeneity. Accordingly, bio-inspired algorithms (BIAs) offer distributed self-organization methods by adapting the natural phenomena such as collaborating bee and ant colonies for the requirements of modern networked systems. In this paper, we leverage ant-colony optimization to solve the joint service allocation and routing (JSAR) problem distributedly, where mixed-criticality services should be distributed and communicated under strict QoS requirements on a virtualized, physical network. We also introduce an integer linear program (ILP) to find the optimal solution for JSAR that can be computed by a centralized controller. Our experiments show that our heuristics successfully solve JSAR for even scaling scenarios. Besides, utilizing different redeployment strategies, they can be adapted to obtain near-optimal results in terms of resource efficiency.**

***Index Terms*—service allocation, routing, bio-inspired, ACO**

I. Introduction

Modern mission-critical systems (MCSs) such as aircraft and industrial facilities have become more complex with numerous inter-connected services and functions. To cope with this complexity, new design paradigms such as virtualization and service-oriented architecture (SOA) are developed. The virtualization of services enables further flexibility for such systems, and is being currently embraced in different domains like avionics [1] and automotive [2]. It allows hosting multiple functions on a single physical node by ensuring isolation and performance guarantees via virtualization [3].

However, SOA also requires the deployment of virtual services and their inter-communication within limited resources and tight quality of service (QoS) requirements. In different areas such as software-defined networking (SDN) and cloud computing, such service allocation and routing problems are handled by centralized controllers that have network-wide visibility [4]. However, they have limited scalability and introduce a single point of failure and thus safety risks. Therefore, decentralized and distributed management techniques are required for further self-organization and reconfiguration capabilities.

Accordingly, bio-inspired algorithms (BIAs) are employed in various domains for distributed task allocation [5], [6]. They adapt the colony behavior of different animal species such as ants and bees that collaborate in nature via certain communication patterns to solve domain-specific problems. In this work, we utilize the ant colony optimization (ACO) for distributed service allocation and routing in virtualized MCSs. Besides, we present an optimization model representing a centralized controller to compare the centralized and distributed solutions. Our contributions are as follows:

- We propose bio-inspired heuristics for distributed service allocation and routing utilizing the ACO.
- We propose an integer linear programming (ILP) optimization model to solve the joint service allocation and routing (JSAR) problem, which can be utilized by a centralized controller. For this, we modified our previous optimization model first presented in [3] by implementing a new fairness multi-objective function and extending the service distribution constraints.
- We compare the efficiency and performance of our distributed bio-inspired heuristics and centralized optimal solution as well as discuss the scalability and the overhead of our heuristics.

The rest of the paper is organized as follows. Section II presents the related work. Section III presents the JSAR optimization model. In Section IV, we give the details of our distributed bio-inspired heuristics. Section V presents the experiment results and Section VI concludes the paper.

II. Related Work

In this section, we briefly present the related work on BIAs and methods for distributed service allocation and routing.

Bio-inspired approaches in networking. BIAs are used in various networking problems that require self-organization without the availability of a centralized management [6], [7]. In this work, we specifically utilize the ant colony optimization (ACO) algorithm [8] as it introduces certain flexibility for designing task distribution and routing problems. In nature, ants leave their pheromone on the trail they travel. A higher pheromone intensity is an indicator that the respective trail is used by more ants and thus preferred by ants, as a well-explored path. Accordingly, without any centralized intelligence, ants can individually find their paths to perform certain tasks. Such an approach is analytically modeled by defining (i) threshold functions to update the level of pheromone per trail, (ii) probabilistic selection functions to select among the available options, e.g., to decide a trail with a certain level of pheromones, and (iii) various design parameters to tune given functions. In networking, [9] utilizes those functions for routing, where increasing pheromone implies the shortest paths. In [10], the authors adapt ACO for energy-efficient routing in wireless networks. The authors of [11] use ACO for coordination and pathfinding for UAVs. Extending routing approaches, [5] first utilize a set of ACO functions for task distribution among robotic agents in a sensor/actuator network and then another set to establish communication between them.

Distributed service allocation and routing. Various techniques are proposed for the distributed service allocation problem. In [12], the authors propose greedy heuristics that can be run by each node individually to improve QoS for service coordination and routing. The authors of [13] leverage the particle swarm optimization to improve the performance of their distributed heuristics to tackle with the same problem. In [14], the authors propose a MILP to model hierarchical and decentralized service distribution and routing framework. Instead of a fully self-coordinated approach, they divide a network into multiple autonomous parts to be organized individually. Another approach in [15] uses machine learning by employing reinforcement learning agents to every node in the network; however, it still requires training those agents offline in a centralized manner. In comparison to the related work, we propose a new problem statement for MCSs and compare the optimal centralized solution with our ACO-based distributed solution reflecting the QoS requirements of the services with mixed-criticality.

III. Optimal Joint Service Allocation and Routing

In this section, we present our joint service allocation and routing (JSAR) model to find the optimal service deployment via a centralized controller. We then use JSAR to compare the optimal solution with the results of our distributed bio-inspired heuristics. We modify our previous model presented in [3] to distinguish between critical and non-critical services and to include additional constraints and a new objective function for fairness in resource utilization. In this sense, the optimal scheme provides a deployment that (i) satisfies all QoS requirements of mixed-critical services within limited system resources and (ii) minimizes the deviation in resource utilization between distinct nodes and links.

Fig. 1 illustrates the JSAR. In the figure, black nodes represent the connected services $s \in S$ in a service overlay. Each service has a binary criticality indicator o_s and certain resource requirements τ_s. Services have to be placed to the physical nodes $u, v \in V$ within their limited resources r_u. Besides, routing should be performed between the host nodes (grey nodes) satisfying given QoS requirements of the communication demands $d \in D$. Each demand is defined between two services, and requires a certain bandwidth h_d for data traffic and has a maximum latency requirement l_d. They should be placed to assigned to the end-to-end paths $p \in P$, which consist of edges $e \in p$ with a limited bandwidth capacity c_e and induced delay l^*_e.

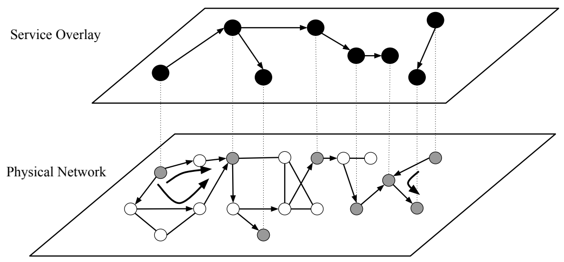


Fig. 1: Service overlay on top of the underlay physical network [3]

$$\min\left\{ \max\left\{\epsilon_l \sum_{d\in D} \sum_{\substack{p\in P,\\ e\in p}} x_{dp} h_d : \forall e \in E\right\} + \max\left\{\epsilon_n \sum_{s\in S} y_{su} \tau_s (1 + o_s(\lambda - 1)) : \forall u \in V\right\}\right\} \quad (1)$$

$$\sum_{u\in V} y_{su} = 1 \qquad \forall s \in S \quad (2)$$

$$\sum_{s\in S} y_{su} \tau_s (1 + o_s(\lambda - 1)) \le r_v \qquad \forall u \in V \quad (3)$$

$$x_{dp} \le y_{sv} y_{tu} + y_{tv} y_{su} \qquad \forall d \in D, \forall u, v \in V, \forall p \in P_{uv}, (s,t) \in d \quad (4)$$

$$\sum_{d\in D} \sum_{\substack{p\in P,\\ e\in p}} x_{dp} h_d \le c_e \qquad \forall e \in E \quad (5)$$

$$\sum_{e\in p} x_{dp} l^*_e \le l_d \qquad \forall d \in D, \forall p \in P \quad (6)$$

$$\sum_{p\in P} x_{dp} \ge 1 \qquad \forall d \in D \quad (7)$$

We introduce two binary decision variables x_{dp} and y_{sv} that represent if demand d is assigned to path p and if service s is deployed on node v, respectively. The multi-objective function (1) ensures fairness by minimizing the maximum link utilization and node resource consumption. Such an objective (i) balances the resource use, (ii) promotes further configurability in case of continuous deployment of new services and demands, and (iii) reduces the risk of a single point of failure that may happen many services and demands are deployed on the same network elements. ϵ_l and ϵ_n are the weighting parameters for two objectives and set to $\epsilon_l = \epsilon_n = 0.5$ as both link and node resource capacity values are adjusted in a similar numerical interval.

Constraint (2) ensures the service s is deployed on exactly one node. Constraint (3) restricts the number of services deployed on node u to guarantee that their total resource requirements do not exceed the node resource r_u. Here, the resource consumption τ_s of critical services (s.t. s is critical if $o_s = 1$) is weighted by factor $\lambda > 1.0$ to avoid deployment of too many critical services on a single node, that can induce single point of failures, performance bottlenecks, or poor resource isolation. Constraint (4) restricts a demand d is assigned on path p only if the required services s and t are deployed on the source and destination nodes of path p, i.e., u and v. Constraint (5) ensures that each link e of path p has sufficient resources c_e to carry the load of demand d. Constraint (6) ensures a path p satisfies the maximum tolerable latency l_d for d. Here, l^*_e represents the latency induced by edge and the end-to-end latency of p is the accumulated delay after each link $e \in p$. Lastly, constraint (7) guarantees that d is assigned exactly to one path.

Constraint (4) is quadratic and linearized using McCormick envelopes [16] as described in [3] to make the overall problem solvable by the state-of-the-art linear optimization tools. The non-linear multi-objective function is also adapted as follows: We define an auxiliary variable z with an additional constraint $z \geq \sum_{d\in D}\sum_{p\in P, e\in p} x_{dp}h_d \quad \forall e \in E$, i.e., greater than the total utilization of every link e. Similarly, we introduce for another variable q to bound the resource consumption. Then, the objective becomes *minimizing* $\epsilon_l z + \epsilon_n q$ to minimize the upper-bound of the utilization per-link and per-node simultaneously. Consequently, it introduces $|E|+|V|$ extra constraints. Further discussion on the complexity of the model can be found in [3].

IV. Bio-inspired Distributed JSAR

In this section, we present our bio-inspired service allocation and routing heuristics. We perform those two parts consecutively together with a redeployment phase. Accordingly, our heuristics have three main processes. First, we assume that all nodes are initialized with the information on available services and inter-service communication demands. Nodes start the *service allocation* process by selecting a set of services probabilistically within their available resources utilizing ACO. They then broadcast their choices and all services are deployed in multiple iterations until they reach a consensus on the overall service deployment. Secondly, according to the given deployment, they initiate the *path discovery and routing* process leveraging a modified version of the distance-vector routing algorithm and another adaptation of the ACO functions. Note that each node utilizes two ACO functions until then: (i) the first function calculates the probability to deploy a particular service, i.e., service probability, and (ii) the second function is to select a forwarding link probabilistically for a particular demand. As the service allocation is performed before routing, a deployment may render inter-service routing infeasible due to the violated QoS constraints or a lack of sufficient capacity on the particular paths. To overcome that, thirdly, we also design a *redeployment* phase, in which the respective services of the violated demands are placed at the different nodes following the same distribution procedure until all demands are assigned.

Similar to JSAR model, we consider the safety requirements of MCSs by (i) avoiding the deployment of several critical services to the same node physical node and (ii) load-balancing the traffic load to widen the demand assignment instead of relying on dedicated shortest paths. Both prevent a system from a single point of failure and performance bottlenecks, which may cause disruptions in the critical services and traffic.

A. Service allocation

We introduce two constraints for service allocation: A node can host (i) a limited number of services within its resource capacity and (ii) a limited number of critical services. We assume that initially, each node obtains a list of services S and demands D from a controller unit, which does not maintain or control the system later on. Each node decides which services it requests according to the function P_u given in Eq. 8.

$$P_u(s) = \frac{\mu_u(s)^\beta}{\sum_{t\in S}\mu_u(t)^\beta} \tag{8}$$

Here, $\beta \in [0, 1]$ is the ACO learning parameter that adjusts the exploration behavior of the ACO heuristic and increases the probability of service to be deployed. The threshold function $\mu_u(s)$ of ACO is calculated as

$$\mu_u(s) = \begin{cases} 0, & \text{if } r^*_u < \tau_s \\ \max\{r_v - \sum_{t\in S_u} y_{tu}(\lambda o_t \tau_t + (1 - o_t)\tau_t), 0\} \end{cases} \tag{9}$$

Note that λ in Eq. 9 represents the weighting factor for the critical services, s.t. $o_s = 1$. It penalizes the deployment of an excessive number of services proportional to their resource consumption. Each node calculates a deployment probability starting from the critical services. The increasing resource consumption here is reversely proportional to the increasing pheromone level in ACO.

After each node u has evaluated the probability to host a service s, $P_u(s)$, for every service, they broadcast the set of services they decide to host together with the computed result of $P_u(s)$. When there is more than one candidate node for the same service, the one with the highest probability gets the respective service. Meanwhile, the nodes also detect the remaining unclaimed services. For the following iterations of the service allocation, hosts announce which of the remaining services they can host additionally similar to the first iteration.

B. Routing

For routing, we modify the traditional distance-vector routing (DVR). DVR requires only local information sharing and recording that directly fits our needs to compute the best next-hop for distributed routing decisions. Here, we extend this mechanism not only to find the shortest path but obtain *a shortest path per adjacent neighbor* so that the node can probabilistically select among a set of suitable paths.

Eq. 10 calculates the shortest distance between u and v in DVR, where $C(u, i)$ defines the cost between nodes u and i on a particular link and corresponds with the parameter l^*_e of JSAR in Section III. N_u represents the set of neighbors of u.

$$D_u(v) = \min\{C(u, i) + D_i(v) \ \ \forall i \in N_u\} \tag{10}$$

Although this approach provides the shortest paths in terms of the given cost function, it is not convenient to assign a higher number of flows to a single shortest path due to resource constraints. Therefore, we extend the potential next-hops by calculating the shortest paths through each neighbor i of a node u to be then selected probabilistically conforming resource and QoS requirements of different flows. Accordingly, Eq. 11 calculates the cost from u to v via next-hop i and it is recorded by v for each $i\ inN_u$ in its routing table instead of keeping only the next-hop providing the shortest path.

$$D^i_u(v) = C(u,i) + D_i(v) \tag{11}$$

We utilize the given distance functions to calculate the probability to assign a flow demand d from v to u via next-hop i within ACO as Eq. 12

$$P^i_{uv}(d) = \frac{r^i_{uv}(d)^\alpha}{\sum_{j\in N_u} r^j_{uv}(d)^\alpha} \tag{12}$$

where α is the ACO learning parameter and $r^i_{uv}(d)$ is the ACO threshold function calculated as

$$r^i_{uv}(d) = \begin{cases} 0, & \text{if } D^i_u(v) > l_d \text{ or } c^*_e < h_d \\ \frac{\gamma_1 D_u(v)}{D^i_u(v)} + \frac{\gamma_2 c^*_e}{c_e}, & \text{otherwise} \end{cases} \tag{13}$$

where c^*_e represents the available resources on link $e \in (u,v)$ and $\gamma_{1,2}$ are the weights s.t. $\gamma_1 + \gamma_2 = 1.0$ to adjust the impact of the cost and resource utilization. Note that the link with higher available bandwidth and leading to the path with a lower cost has higher probability according to Eq. 13. Therefore, the link utilization is reversely proportional with the pheromone level in ACO.

This process only provides *best-case cost estimation* for the latency but does not guarantee the utilization of the shortest one as each node selects the next hops probabilistically in a distributed manner. If a node receives the same packet twice, it directly bounces it back to the sender node together with a notification message so that the sender node can re-route it through another link to avoid loops.

C. Redeployment

After the deployment, some demands may not be satisfied since (i) the distance between the nodes u and v hosting respective services s and t, e.g., in terms of number of hops, might be too far to satisfy the required QoS or (ii) there might be insufficient bandwidth on the available paths between those two nodes. In such cases, we follow one of those strategies shown in Fig. 2 for the service and demand redistribution: (i) deploying duplicate service instances to satisfy the remaining demands or (ii) migrate the respective services of unsatisfied demands and embed the service overlay as it is.

Extra service deployment. In Fig. 2a, s_1, s_2 and s_3 should communicate and s_1 and s_3 are deployed on the given nodes. If s_3 does not necessarily use the data originated from s_1 and processed at s_2, two duplicate instances of s_2 can be placed to satisfy the inter-service demands separately. For such cases, we propose *bia-flex* that deploys duplicate service instances

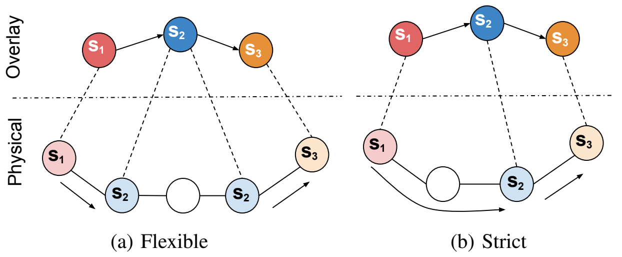


Fig. 2: Different redeployment strategies

flexibly to satisfy the different demands that utilize the same services. Consequently, both demands between $s_1 - s_2$ and $s_2 - s_3$ can be assigned to the shortest paths possible without potentially violating any QoS requirements, e.g., they induce only 1-hop latency in this scenario. Accordingly, a similar service allocation and routing processes (cf. Section IV) are repeated only for the remaining demands and extra service instances in each redeployment iteration.

bia-flex loosens the assumption that services in the given overlay are tightly dependent on each in a given order and extends the solution space. However, it also costs extra node resources to host redundant services. To avoid that, we developed another strategy, where each service is only deployed once and reused by all demands utilizing it.

Strict redistribution. Fig. 2b illustrates the second redeployment strategy, *bia-strict*. It is analogous to the optimal distribution presented in Section III and deploy the service overlay as a whole assuming that each service *strictly* requires the data coming from the preceding service to process and send to the successor service. In the figure, s_2 is deployed on a node between the hosts of s_1 and s_3 to receive data from s_1, process them, and forward to the s_3 without losing any contextual information contained in the input of s_1. The challenge is, if the requirements of even only one of the corresponding demands cannot be satisfied, *bia-strict* migrates the service instances *starting from the services that are least common among deployed demands*. In this scenario, s_2 could be moved until both demands between $s_1 - s_2$ and $s_2 - s_3$ are satisfied. Eventually, it triggers several iterations of redeployment until all demands are assigned.

After the migration of a service instance, the threshold function of the node u that currently hosts that service $\mu_u(s)$ (and similarly $\mu_v(t)$) are reduced s.t. $\mu^*_u(s) = \eta\times\mu_u(s)$, where $\eta \in [0,1]$ to avoid the same deployment without completely blocking u to host the same services. Similarly, $r^i_{uv}(d)$ is modified to $r^{i*}_{uv}(d) = \eta \times r^i_{uv}(d)$ to penalize the routes that have lead to an unsatisfying assignment before a migration is triggered. Note that nodes should disseminate the information for failing demands and the migrated services to update the threshold functions of the respective nodes.

V. Evaluation

In this section, we summarize our experiment results. We evaluated (i) the fairness in resource utilization with respect to the increasing size of service overlay in small scenarios, (ii) the scalability of heuristics in terms of resource consumption and QoS, and (iii) the redeployment cost of our heuristics in terms of the number of extra services deployed in *bia-flex* and the number of redeployment iterations. For all experiments, all of the services and demands are successfully embedded in the physical network, i.e., the optimal and heuristic results are given for the 100% deployment ratio. For the network topologies, we generated random networks with an average node degree of 2.5. Service overlays are also randomly generated with similar connectivity. All experiments are repeated 40 times for the heuristics and 20 times for the optimal results.

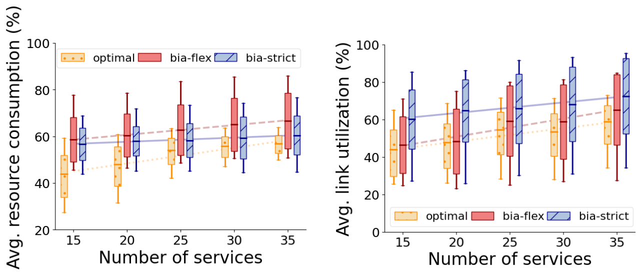
Our heuristics have numerous parameters as we described in Section IV. We set $\beta = 0.8, \alpha = 0.8, \lambda = 1.8, \mu = 0.8, \gamma_1 = 0.3$, $\gamma_2 = 0.7$ and $\mu = 0$ as they give the optimal results regarding Eq. 1 as a result of our sensitivity analysis.

Resource utilization. Fig. 3 shows the node resource utilization and link utilization for an increasing size of service overlay. The box plots for *optimal* (orange, dotted), *bia-flex* (red, solid), and *bia-strict* (blue, hatched) show the (averaged) minimum, maximum (edges of the vertical lines), and mean (mid-line) utilization of the active network components, and the standard deviation (boxes around the mid-line) in resource consumption. While a smaller maximum value and standard deviation indicate a fairer resource usage, the mean utilization shows resource efficiency when increasing the size of the overlay. For those experiments, we set the number of demands more than the number of services s.t. $|D| = |S| + 10$ to ensure that multiple demands utilize the same services, and thus a more challenging service overlay can be embedded into the network. The network size is set to 20 nodes.

In both Fig. 3a and 3b, the optimal placement results in the minimum deviation and maximum value in resource utilization, which means every node and link used in deployment has a similar amount of load. In Fig. 3a, *bia-flex* leads to a higher utilization of node resources than *bia-strict*. This is mainly because it requires placing extra service instances. The difference in the maximum utilization stems from that the nodes with the highest connectivity, i.e., the most central ones, tend to host the extra service instances more often. The reason is, that after multiple redeployment iterations, most of the nodes still cannot satisfy the QoS requirements of the demands requiring extra service instances, and such high-connectivity nodes provide the most suitable routes. In contrast, *bia-strict* stays closer to the *optimal* regarding the mean utilization.

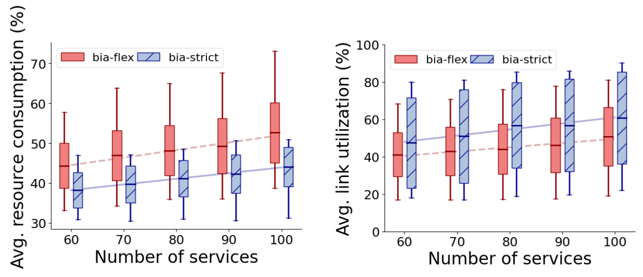
Fig. 3b shows that *bia-strict* results in more congested links. It leads to a denser service and data traffic deployment as all service instances should be connected strictly. In contrast, *bia-flex* utilizes a wider range of links by initiating extra service instances on different nodes. Therefore, it enables the use of alternative routes. Accordingly, for the link utilization, *bia-flex* gives closer results to the optimal solution.

Scalability. Fig. 4 shows the scalability of our heuristics in terms of node and link utilization for larger service overlays. The network size is set $|V| = 50$ and the number of demands



(a) Node resource consumption (b) Link utilization

Fig. 3: Resource utilization for increasing size of service overlay



(a) Node resource consumption (b) Link utilization

Fig. 4: Resource utilization for scaling service overlay

is $|D| = |S| + 20$. Note that the optimization model cannot be solved in a reasonable time due to its complexity and thus only the results of the heuristics are presented here. In both Fig. 4a and Fig. 4b, the results show a similar trend with the ones in the smaller scenarios (cf. Fig. 3). Therefore, regardless of the size, it can be concluded that while *bia-flex* achieves better link utilization in less connected networks, *bia-strict* achieves better results for the networks of low-capacity nodes. Apart from the resource efficiency, Fig. 5 shows the average latency of inter-service communication in terms of the ratio of the average length of allocated routes to the network diameter, which is the longest shortest path in the network. We normalized the length, i.e., the number of hops, to present it as proportional to the network size. Supporting our findings in Fig. 3b and Fig. 4b, *bia-flex* (red, dashed) results in lower latency and better QoS by selecting shorter paths as it can leverage a more flexible service deployment. Note that both *bia-flex* and *bia-strict* satisfy the QoS requirements of all demands strictly; however, *bia-flex* induces better routes.

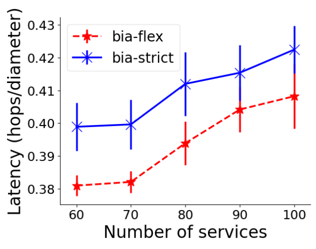
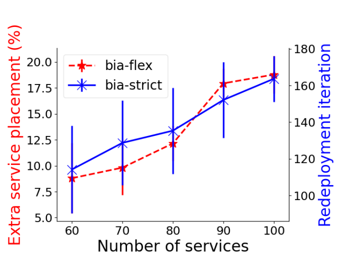
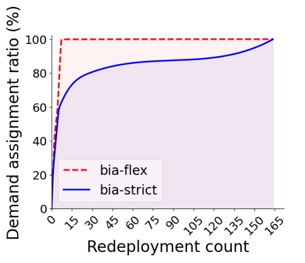


Fig. 5: Latency (normalized with network diameter)



(a) Cost of redeployment strategies



(b) Demand assignment distribution

Fig. 6: Reployment cost in terms of extra services and iterations

Redeployment cost. Fig. 6 shows the redeployment cost in our heuristics in terms of the extra service placement and the redeployment time. In Fig. 6a, we show the percentage of the services that require extra instances for *bia-flex* on the left y-axis (red) and the required number redeployment iterations for *bia-strict* on the right y-axis (blue). Note that *bia-flex* also requires a neglectable number of redeployment iterations, e.g., less than 10 in the given scenarios, and thus it is not shown in the figure. As seen in the figure, while *bia-flex* can require up to 20% extra instances for the largest service overlay, *bia-strict* takes more than 150 redeployment iterations, which induces further convergence time and also data overhead for the consensus between the nodes. Therefore, while the former costs extra node resources, the latter requires a longer time to settle the whole service deployment.

Although a high number of redeployment iterations is required in *bia-strict* for a fully-functional network, both heuristics deploy the majority of the demands rather quickly. Fig. 6b shows the cumulative percentage of the demand assignment by the number of redeployment iterations. *bia-flex* (red, dashed) deploys all services and demands under 10 iterations by placing extra service instances. *bia-strict* can assign 80% of the demands in the first 30 iterations and the rest takes further time. The reason is that the most connected services, i.e., the services utilized by many demands, should be replaced until they satisfy the QoS requirements of *all* respective demands. Initially, the search space, i.e., the available nodes and links, for those is rather large but after some iterations (120 in the figure) it converges much faster as many nodes and links are blacklisted due to QoS violations. Accordingly, it is possible to use *bia-flex* for dynamic or incremental service and demand assignment as it has more than 10 times better convergence time with up to 20% extra resource cost. Both can also be used together to first deploy a fully-functional network with all demands in place and then to shift to the strict embedding of the service overlay in time for better resource efficiency. The optimal solution, on the other hand, takes significantly longer time than the heuristic results even for small scenarios.

VI. Conclusion

Mission-critical systems (MCSs) has started to leverage virtualized service-based architecture for further design flexibility and dynamicity. However, it introduces configuration overhead to efficiently allocate all required services and communication demands. They usually require a centralized controller with network-wide visibility for such configuration, which offers limited scalability and is at the risk of being a single point of failure. In this paper, we propose bio-inspired joint service allocation and routing (JSAR) heuristics for virtualized MCSs to reduce the dependency to a centralized entity. We adapt the analytical framework of the ant-colony optimization technique to distribute mixed-criticality services and establish routes for the data traffic with various QoS requirements. We also introduce an integer linear program (ILP) to find an optimal configuration scheme that can be computed by a centralized controller to compare the performance of the centralized and distributed approaches. Our evaluation shows that the proposed heuristics perform close to the optimal results in terms of resource efficiency and fairness by adapting different redeployment strategies. Our heuristics can be used in different scenarios depending on the available resources, time, or necessity of dynamic service deployment.

References

[1] *Future Airborne Capability Environment (FACE) Technical Standard Edition 3.1.* The Open Group, Jul 2020.
[2] *Automotive Virtual Platform Specification.* GENIVI, Jul 2020.
[3] D. Ergenc, J. Rak, and M. Fischer, "Service-Based Resilience for Embedded IoT Networks," in *50th IEEE/IFIP International Conf. on Dependable Systems and Networks (DSN)*, pp. 540–551, 2020.
[4] R. Mijumbi, J. Serrat, J. Rubio-Loyola, N. Bouten, F. De Turck, and S. Latré, "Dynamic resource management in sdn-based virtualized networks," in *10th Int. Conf. on Network and Service Management (CNSM)*, pp. 412–417, 2014.
[5] T. H. Labella and F. Dressler, "A bio-inspired architecture for division of labour in SANETs," in *Advances in Biologically Inspired Information Systems*, pp. 209–228, Springer, 2007.
[6] F. Dressler and O. B. Akan, "Bio-inspired networking: from theory to practice," *IEEE Comm. Mag.*, vol. 48, no. 11, pp. 176–183, 2010.
[7] C. Zheng and D. C. Sicker, "A survey on biologically inspired algorithms for computer networking," *IEEE Comm. Surveys & Tut.*, vol. 15, pp. 1160–1191, 2013.
[8] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on evolutionary computation*, vol. 1, no. 1, pp. 53–66, 1997.
[9] G. Di Caro, F. Ducatelle, and L. M. Gambardella, "AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks," *European Trans. on Telecom.*, vol. 16, no. 5, pp. 443–455, 2005.
[10] A. S. Sharma and D. S. Kim, "Energy efficient multipath ant colony based routing algorithm for mobile ad hoc networks," *Ad Hoc Networks*, vol. 113, p. 102396, 2021.
[11] Y. He, Q. Zeng, J. Liu, G. Xu, and X. Deng, "Path planning for indoor UAV based on Ant Colony Optimization," in *25th IEEE Chinese Control and Decision Conference (CCDC)*, pp. 2919–2923, 2013.
[12] S. Schneider, L. Dietrich Klenner, and H. Karl, "Every Node for Itself: Fully Distributed Service Coordination," in *16th Int. Conf. on Network and Service Management (CNSM)*, 2020.
[13] A. Song, W.-N. Chen, T. Gu, H. Yuan, S. Kwong, and J. Zhang, "Distributed virtual network embedding system with historical archives and set-based particle swarm optimization," *IEEE Trans. on Systems, Man., and Cybernetics*, vol. 51, no. 2, pp. 927–942, 2021.
[14] S. Schneider, M. Jürgens, and H. Karl, "Divide and Conquer: Hierarchical Network and Service Coordination," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 54–62, 2021.
[15] S. Schneider, H. Qarawlus, and H. Karl, "Distributed Online Service Coordination Using Deep Reinforcement Learning," in *IEEE 41st Int. Conf. on Dist. Computing Systems (ICDCS)*, pp. 539–549, 2021.
[16] G. P. Mccormick, "Computability of Global Sol. to Factorable Nonconvex Programs: Convex Underestimating Problems," *Math. Program.*, vol. 10, pp. 147–175, Dec. 1976.

Appendix D

Moving Target Defense for Service-oriented Mission-critical Networks

Abstract

Modern mission-critical systems (MCS) are increasingly softwarized and connected. As a result their complexity increased and so their vulnerability against cyber-attacks. However, the current adoption of virtualization and service-oriented architectures (SOA) in MCSs provides additional flexibility that can be leveraged to withstand and mitigate attacks, e.g., by moving critical services or data flows. This enables the deployment of strategies for moving target defense (MTD), which allows to strip attackers from their asymmetric advantage from long-reconnaissance of MCSs. However, it is challenging to design such an MTD mechanism, given the diverse threat landscape, resource limitations, and without degrading the availability of services. In this paper, we combine different optimization models to explore feasible service distributions and routing configurations for SOA-based systems and to derive optimal MTD sequences and schedules, i.e., to decide on subsequent system configurations and on their timing, based on an attacker-defender game. Our results indicate that even for challenging and diverse attack scenarios, it is possible to defend the system by up to 90% of the system operation time with a limited MTD defender budget.

Reference

Doğanalp Ergenç, F. Schneider, P. Kling, M. Fischer. Moving Target Defense for Service-oriented Mission-critical Networks. International Conference on Computer Communications and Networks (ICCCN), 2023.

Contribution

In the forementioned publication, the main contribution belongs to this thesis. The second co-author contributed to writing preliminaries explaining an existing game-theoretical model. All co-authors also helped to improve the quality of the paper with their valuable feedback.

Moving Target Defense for Service-oriented Mission-critical Networks

Doğanalp Ergenç, Florian Schneider, Peter Kling, Mathias Fischer
Universität Hamburg, DE
name.surname@uni-hamburg.de

***Abstract*—Modern mission-critical systems (MCS) are increasingly softwarized and interconnected. As a result, their complexity increased, and so their vulnerability against cyber-attacks. The current adoption of virtualization and service-oriented architectures (SOA) in MCSs provides additional flexibility that can be leveraged to withstand and mitigate attacks, e.g., by moving critical services or data flows. This enables the deployment of strategies for moving target defense (MTD), which allows stripping attackers of their asymmetric advantage from the long reconnaissance of MCSs. However, it is challenging to design MTD strategies, given the diverse threat landscape, resource limitations, and potential degradation in service availability. In this paper, we combine two optimization models to explore feasible service configurations for SOA-based systems and to derive subsequent MTD actions with their time schedule based on an attacker-defender game. Our results indicate that even for challenging and diverse attack scenarios, our models can defend the system by up to 90% of the system operation time with a limited MTD defender budget.**

***Index Terms*—moving target defense, game theory, service-oriented architecture**

I. Introduction

Modern mission-critical systems (MCSs), like smart cars and avionics, consist of interconnected services that carry out collaborative tasks. This results in additional complexity and thus, a broader surface for cyber-attacks. To cope with the additional complexity, service-oriented architectures (SOA) and virtualization are increasingly adopted in different mission-critical domains [1]–[3]. SOA can accommodate the system design by enabling flexible and isolated service deployment on virtualized hardware. Such flexibility also enables a reconfiguration of systems to handle failures and to withstand and recover from cyber-attacks.

From a security perspective, attackers have an asymmetric advantage against traditional MCSs since they can conduct a long reconnaissance before they carry out their attacks [4]. Besides, an attacker can remain in stealth for months to make the highest impact even after infiltrating a system [5]. Here, the longer the system remains in its static configuration, the higher the probability of a successful attack is. Defenders, however, have only a limited time to detect and mitigate it. Moving target defense (MTD) can balance this asymmetry by reconfiguring critical assets [6], e.g., shuffling IP addresses or changing the allocation of critical services. It renders the attacker's knowledge about the system obsolete and thus impedes attacks. SOA and virtualization ease the development of MTD strategies as they enable the migration and replacement of services and reconfiguration of their inter-communication.

However, MTD via service reconfiguration requires additional spare resources and induces reconfiguration costs for increased delay and packet loss. Furthermore, without a precise understanding of potential attacks and failures, an MTD strategy causes too frequent or ineffective reconfigurations [7]–[9]. Therefore, we need an effective MTD strategy that determines *which* services must be changed, *how* they are changed (e.g., migrate or re-instantiate), and *when* they are changed. To address those questions, various attacker-defender games have been proposed in the context of game theory, e.g., FlipIT [10] or the probabilistic learning attacker and dynamic defender (PLADD) model [11]. Although they have already derived asymptotical bounds for optimal MTD strategies, they do not provide concrete steps to reconfigure systems. Moreover, these models do not include network design constraints for resource management and quality of service, which is especially important for MCSs.

This paper proposes an optimization framework to determine subsequent service configurations within optimal MTD strategies based on an attacker-defender game. Accordingly, our contributions are:

- We repurpose our linear programming model for joint service allocation and routing (JSAR) [1], [12] to identify a set of feasible service configurations satisfying resource and QoS requirements of SOA-based MCSs.
- We formulate a novel optimization model, PLADD-scheduling (PLSCH) based-on the PLADD game [11], to find optimal MTD schedules against various attacks.
- We develop a composite model, PLSCH-MTD, to deploy the resulting configurations of JSAR for each MTD action over the time-schedule provided by PLSCH.
- We create several attack scenarios reflecting the time characteristics of recent security incidents in MCSs to evaluate PLSCH-MTD.

In the rest of the paper, Section II introduces the preliminaries for the considered attack-defender game. Section III presents related work on SOA design and MTD. Section IV introduces our optimization models JSAR, PLSCH, and PLSCH-MTD. Section V describes the attack scenarios that are used to evaluate PLSCH-MTD. Section VI presents the evaluation results, and Section VII concludes the paper.

II. Background

In this section, we describe two essential concepts of this study: The probabilistic learning attacker and dynamic defender (PLADD) model and the PLADD-scheduling problem. We also note our assumptions and modifications that make the formulation of these models more convenient for this study.

A. Probabilistic Learning Attacker and Dynamic Defender

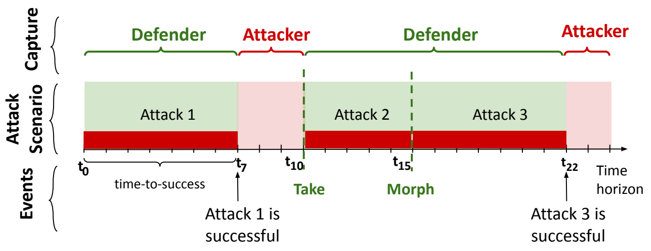


Fig. 1: An example of the PLADD game.

PLADD introduces an attacker-defender game that involves (i) an attacker with learning capabilities and (ii) a defender with various actions competing to gain control of the system within a given *time horizon* that represents a certain frame of the system's operation time [11]. Fig. 1 shows its fundamentals. An attacker can conduct successive attacks (red blocks) that each takes a certain time to be completed, i.e., having *time-to-success*. As a result of a successful attack, the attacker captures the resources (indicated by the light red background, e.g. from t_7 to t_{10}). When an attacker completes an attack, it might *learn* about the system, and its subsequent attack takes less time accordingly, e.g., attack 2 is shorter than attack 1.

The role of a defender is to conduct certain actions (vertical dashed lines) to prevent an attacker from completing its attack. A *take* action usually represents an instant intervention, e.g., resetting a service instance, while a *morph* action refers to more substantial system changes, e.g., migrating multiple services over the system nodes with diverse configurations. After the defender morphs the system, the attacker loses her knowledge obtained after successful attacks and thus should spend a longer time for its upcoming attacks (e.g., attack 3 in Fig. 1 is longer than attack 2). Similar to the *take* action, the defender captures the resources back after *morph*.

Both an attacker and a defender have limited *budget*. An attacker can have only limited attacking opportunities, and lengthy attacks require more effort. A defender cannot reconfigure the system too often, and the cost of a reconfiguration is usually proportional to the changes in the system, as they usually cause service interruptions. Therefore, the defender should conduct its actions within an effective time-schedule against potential attacks within its budget.

Eventually, in the PLADD model, the goal of the attacker is to complete a sequence of attacks and gain control over the system after each successful attack. The defender aims to develop a strategy that determines (i) the type of defensive action to prevent an attack and (ii) a schedule to conduct a sequence of actions against particular attack scenarios within its limited budget.

We mainly focus on PLADD as it can model different types of MTD actions and their effective scheduling to minimize the attacker's advantage. Furthermore, it can capture the time characteristics of several attacks, which can vary from relatively fast reconnaissance attempts to long-term advanced persistence threats (APTs) in MCSs. In our formulation, we assume that the time-to-success of an attack is independent of the previous successful attacks, i.e., the attacker does not learn. It enables us to develop defensive strategies against potential attack scenarios, whose characteristics can be modeled in advance. As a result, we consider a single type of defensive action (*take* or *morph*), referred to as the *infinite* model in [11]. This action corresponds to *morph* in the original PLADD model regarding its impact since the service migrations over the system lead to a significant reconfiguration. Lastly, we have not limited the attacker to a certain budget and assume that it can conduct attacks whenever the defender regains control over the system.

B. PLADD-Scheduling

PLADD-Scheduling (PLSCH) leverages the PLADD game to provide an exact schedule for the defensive strategy, e.g., when to conduct *take* or *morph* actions. Originally proposed in [13], it formulates the attacker-defender game as a combinatorial job assignment problem. Here, we first describe the job assignment problem and then explain how it corresponds to the original PLADD model. It considers a system of $m \in \mathbb{N}$ machines over a time horizon of $T \in \mathbb{R}^+$ time units. Each machine m comes with a job sequence $J_m = (d_{m1}, d_{m2}, \dots)$ of at most $n + 1$ jobs which it must process. The *duration* $d_{mj} > 0$ of the jth job on machine m specifies how long it takes machine m to process its jth job. In order to start processing the jth job of J_m, machine m must have finished the first $j - 1$ jobs. A job on any machine can only start after a *starting action*, which affects all machines simultaneously, is taken. However, the number of those actions is limited and thus, they should be scheduled effectively to initiate several jobs across multiple machines. The time between the end of a job and the beginning of a subsequent job, i.e., if the jobs cannot be scheduled adjacently, is idle machine time.

To see the connection to the PLADD game, we interpret each machine as one of m possible, equally likely attack scenarios. A job j corresponds to an individual attack within an attack scenario, and its duration is the time-to-success value for the respective attack. A finished job means that the attacker captured the resources. The starting action for the jobs corresponds to the instantaneous time that the defender retakes control of a potentially compromised system, i.e., a *take* action in the PLADD model. The limitation on the number of starting actions represents a limited defender budget. Note that the attacker continuously conducts attacks right after each defender action competing for the system resources.

With this interpretation, the goal in PLSCH becomes to schedule the jobs such that the total idle time over all machines is minimized. This time also corresponds to the duration when the resources are under the attacker's control. Note that minimizing the idle time is equivalent to maximizing the total time any machine is active (not idle).

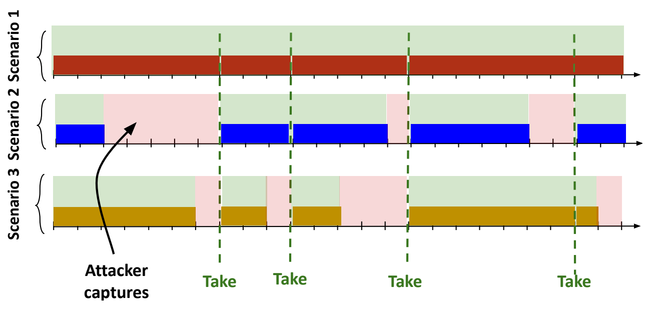


Fig. 2: A single defensive schedule for multiple attack scenarios.

In PLSCH, a single action determines the job assignment on multiple machines. That is, the jobs on all machines initiate simultaneously according to a single schedule of starting actions. The reason lies in the formulation of the PLADD model: The defender cannot know the actual attack scenario and, thus, must develop the most effective strategy to defend against all likely attack scenarios. Accordingly, all the jobs are generated in advance as input to the model, reflecting the potential attack scenarios against the target system. Fig. 2 illustrates the difficulty of scheduling jobs over multiple machines, which corresponds to protecting the system against various attack scenarios simultaneously. While there is no idle time on the first machine, i.e., complete protection against the first attack scenario, the same schedule results in more idle times on the other machines, i.e., resulting in the attacker's success.

III. Related Work

In this section, we present the state-of-the-art on (i) SOA-based network design, (ii) moving target defense, and (iii) game-theoretical approaches for network security.

Service Distribution and Network Design: In SOA-based mission-critical networks, the critical services and flows are the assets to be protected. Therefore, a reconfiguration in the context of MTD involves a service allocation and flow assignment problem. A proper service allocation [14], [15] is important to, for instance, minimize operational costs [16] and physical resource fragmentation [17] for the providers, and maximize the service quality [16] and responsiveness [18] for the user experience. It usually requires an accurate resource orchestration regarding where, when, and how many service instances are deployed [19], [20]. Besides, the dependencies of services on each other [21], service migrations [22], load-balancing [23], task scheduling [24], and power-awareness [25] are some of the design constraints that are addressed in the literature. Other studies address the service allocation and routing problem jointly to deploy the services on the paths aiming for optimal resource utilization [26], [27]. Recent studies include the service protection and availability issues as well [1], [12], [28].

In this work, we use our previous service allocation and flow assignment scheme [1] as it directly reflects the SOA requirements of the MCSs. Moreover, it offers configurability with dynamic services and flows, which gives a large reconfiguration space for potential MTD strategies.

Moving Target Defense: MTD is a well-studied field that enables the development of defensive strategies by moving the critical assets in a system. The authors of [29] implement multiple diverse platforms with different software packages, operating systems, and processor architectures. The system functions are then moved among such platforms keeping the state information. Similarly, in [30], a pool of diverse virtual systems is orchestrated by a controller to fluctuate the attack surface by switching on and off the redundant resources on different components. In [31], it is argued that any configuration parameter may impact the overall security. They propose a genetic algorithm to find the best suitable (re)configuration to minimize the chance of a successful attack. The authors of [32] focus on mutating the network configuration, e.g., IP addresses, ports, and destination addresses. In [8] and [9], they circulate the virtual machines with different operating systems as well as change network addressing schemes to prevent both OS- and network-targeted persistent attacks.

In contrast to the related work, we consider the services and flows as our critical assets in the SOA context. Although several studies merely focus on migrating virtual instances, we propose an optimization framework to find feasible and timely system-wide reconfiguration.

Game theory: Game theory offers solid analytical tools to develop attacker and defender interactions to develop effective defensive strategies [33], [34]. The same authors of PLADD extend their evaluation with further insights in [35]. According to the practical implications of the study, it is always possible to push a rational attacker out of the PLADD game even though it might not be cost-optimal for the defender. PLADD has also been considered for the security modeling of various networking areas. In [36], the authors utilize PLADD to defend power grid infrastructure. They analyze the optimal schedule to reset access controls of the system to minimize the probability of a successful attack. In [37], the authors focus on the multi-attacker and defender games for massive machine-type communications (mMTC) in 5G. They formulate a non-zero-sum differential game with attack and defense alliances and propose an optimal defensive strategy algorithm. In [38], the authors address APTs toward cloud systems. In this game model, two parties compete to set their attack and scan intervals based on their subjective decisions. In [39] and [40], the authors formulate spatio-temporal Stackelberg games to find optimal configurations for web applications over time.

In comparison to the related work, we model various attack scenarios in terms of their time characteristics against MCSs rather than focusing on smaller-scale web applications. We also evaluate different types of attacks beyond specific vulnerabilities of their target applications. In addition, we address the complex interdependencies of connected services regarding their resource consumption and QoS in SOA-based MCSs.

IV. PLADD-Scheduling MTD (PLSCH-MTD) Optimization Model

A. Solution Overview

Fig. 3 shows the steps of the overall model, PLSCH-MTD, which consists of two optimization processes: (i) Joint Service Allocation and Routing (JSAR) and (ii) PLADD-Scheduling (PLSCH). While JSAR provides the possible configurations to be used within MTD actions (green blocks), PLSCH finds a schedule for the defender's actions by changing respective configurations to defend against the considered attack scenarios (red blocks). In the rest of this section, we discuss, first, the PLSCH process for scheduling and then present the integration of MTD to the given model accordingly.

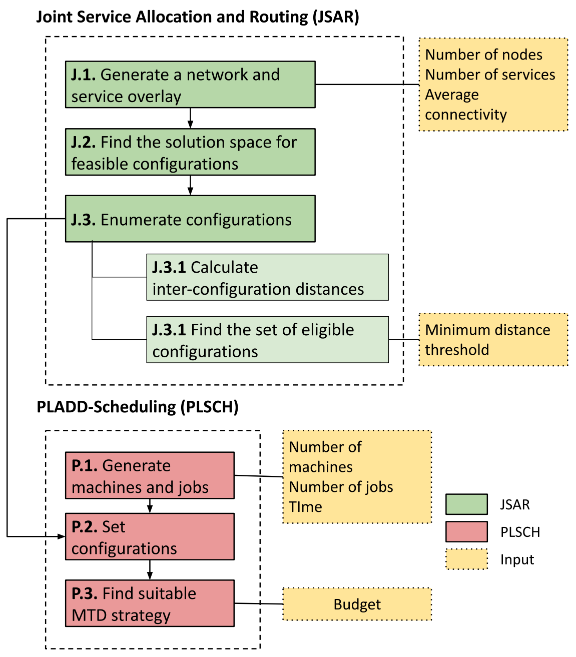

Joint Service Allocation and Routing (JSAR)
J.1. Generate a network and service overlay
J.2. Find the solution space for feasible configurations
J.3. Enumerate configurations
J.3.1 Calculate inter-configuration distances
J.3.1 Find the set of eligible configurations
Number of nodes Number of services Average connectivity
Minimum distance threshold
PLADD-Scheduling (PLSCH)
P.1. Generate machines and jobs
P.2. Set configurations
P.3. Find suitable MTD strategy
Number of machines Number of jobs TIme
Budget
JSAR
PLSCH
Input

Fig. 3: The overall optimization framework: PLSCH-MTD.

B. PLADD-Scheduling (PLSCH) Model

In this section, we formulate the PLSCH problem described in Section II-B as an integer linear program (ILP). The idea of such combinatorial model is introduced in [13] and we modify and extend the model by implementing time-to-success requirements and with further constraints. Table I shows all related variables and parameters.

TABLE I: Variables and parameters of PLSCH.

Type	Symbol	Set	Definition
Base	m	M	A machine
	j,k	J_m	A job to be scheduled on machine m
	t,u	Z^*	Discrete time instance
Constant	d_{mj}	Z^*	Duration of job j on machine m
	β	Z^*	Defender budget
Variable	x_t	Z^*	Decides if an action taken at t
	y_{mjt}	Z^*	Decides if j scheduled on m at t

As explained in Section II-B, we represent each attack scenario as a *machine* and individual attacks in each scenario as *jobs* in the PLSCH model. The PLSCH takes (i) a number of machines with different sequences of jobs and (ii) a fixed action budget as input. It provides a schedule for the multiple-machine job assignment problem that corresponds an MTD schedule for the defender. Accordingly, there are two optimization variables, x_t and y_{mjt}. x_t is a binary decision variable that represents if any starting action is schedule at the time instance $t < T$ to initiate a job, where T is the system's operational time, i.e., time horizon. y_{mjt} is the other binary variable to decide if a job $j \in J_m$ of machine $m \in M$ is scheduled to start at the time instance t. Note that the number of starting actions are limited by the action budget, which corresponds to the defender budget in PLADD. All other constraints are given as follows.

$$\sum_{t=0}^{T} y_{mjt} \leq 1 \qquad \forall m \in M, \forall j \in J_m \tag{1}$$

Constraint 1 ensures that j can be scheduled only once on m.

$$\sum_{t=0}^{T} y_{mjt} - \sum_{t=0}^{T} y_{mkt} \geq 0 \quad \forall m \in M, \forall j,k \in J_m, k = j+1 \tag{2}$$

Constraint 2 ensures that a job k can take place on machine m only if its predecessor job j, i.e., $k = j+1$, is scheduled on m at a time instance t. It implies that a job k cannot be scheduled on a machine m before all other jobs j in the job set J_m s.t., $j < k$ are placed. Accordingly, all attacks in each attack scenario are defended in the given order.

$$(d_{mj} + t)y_{mjt} + (T - u)y_{mku} \leq T$$
$$\forall m \in M, \forall j,k \in J_m, k = j+1, \forall t,u \leq T \tag{3}$$

Constraint 3 ensures that two consecutive jobs j and k s.t. $k = j+1$ in a single attack scenario cannot overlap as the successor job k restricted to start after the whole duration of j, d_{mj}, s.t. $t + d_{mj} \leq u$, where t and u are the starting times of j and k, respectively. Besides, the finishing time of j is constrained by the total operating time of the system, T, in case there is not successor job scheduled. Note that non-overlapping jobs in a machine in PLSCH formulation could imply that there cannot be concurrent attacks in an attack scenario in PLADD game. However, the PLSCH handles that by introducing multiple machines so that the resulting strategy can defend against multiple attack scenarios simultaneously.

$$\sum_{j \in J_m} y_{mjt} \leq x_t \qquad \forall m \in M, \forall t \leq T \tag{4}$$

$$\sum_{m \in M} \sum_{j \in J_m} y_{mjt} \geq x_t \qquad \forall t \leq T \tag{5}$$

Constraints 4 and 5 represent the dependencies between two decision variables. Constraint 4 ensures that (i) no job j can be scheduled at t unless there is a starting action s.t., $x_t = 1$ and (ii) at most one job can be placed on machine m at a given time instance t. Complementarily, constraint 5 implies that there should be at least a job scheduled in one of the machines if a starting action takes place at the given time. Those constraints also model the dynamics of the PLADD game, s.t., an attacker is expected to conduct a new attack right after the defender takes an action and regains the control.

$$\sum_{t=0}^{T} x_t \leq \beta \tag{6}$$

Lastly, constraint 6 limits the number of starting actions by β. In PLADD, it corresponds to the limited defender budget in terms of the number of MTD actions.

The objective function 7 maximizes the occupation of a machine with respective jobs. This corresponds to the time spent by the attacker to conduct attacks when the defender holds control of the system. It also implies the minimization of the idle time of all the machines, i.e., decrease the time when the attacker captures the system [13]. Therefore, it eventually aims to protect the system from being occupied by the attacker considering all (given) potential attack scenarios.

$$\max \sum_{m \in M} \sum_{j \in J_m} \sum_{t=0}^{T} y_{mjt} d_{mj} \tag{7}$$

C. JSAR: Network Configuration Model

Besides scheduling MTD actions, the defender must decide which configuration to apply to take adequate measures. A configuration consists of (i) allocating mixed-criticality services over virtualized MCS nodes and (ii) establishing their intercommunication within limited system resources. This decision is highly dependent on the structure of the network. In [1], we proposed JSAR as an optimization model for the design of mission-critical networks according to the given definition. Here, we utilize the model to generate a feasible solution space, e.g., a set of sub-optimal configurations, that can be used by the defender to change the deployment of the network. We present the details of JSAR in this section.

The JSAR takes (i) a network of nodes with different processing capacities and connected via links with limited bandwidth and (ii) a service overlay with inter-connected services with certain QoS demands. z_{dp} and q_{sv} are two binary decision variables that represent if demand d is assigned to path p and if service s is deployed on node v, respectively. The objective function (8) minimizes the length of selected paths, where $|p|$ represents the path length. Minimizing the total path length can be considered as both performance and cost optimization by establishing low-latency communications, i.e., here with fewer hops, and decreasing the number of occupied links. Depending on the various goals of the defender as a network designer, the objective function can be easily adapted.

Constraint (9) and (10) ensure that v has sufficient resources to host s and s is deployed on exactly one node that is capable to host s (e.g., equipped with the required hardware). Constraint (11) restricts that d can be deployed on p only if the required services s and t are deployed on the source and destination nodes of path p, which are u and v. This quadratic constraint is linearized using McCormick envelopes [41]. Constraint (12) ensures that each link e of p has sufficient resources to carry the traffic of d if it is assigned to p. While constraint (13) ensures that p is selected to satisfy the maximum tolerable latency for d, constraint (14) guarantees that d is assigned exactly to one path.

$$\min \sum_{d \in D} \sum_{p \in P} z_{dp} |p| \tag{8}$$

$$\sum_{s \in S} q_{sv} r_s \leq r_v \qquad \forall v \in V \tag{9}$$

$$\sum_{v \in V} o_{sv} q_{sv} = 1 \qquad \forall s \in S \tag{10}$$

$$z_{dp} \leq q_{sv} q_{tu} + q_{tv} q_{su} \qquad \forall d \in D, \forall u, v \in V, \forall p \in P_{uv}, (s,t) \in d \tag{11}$$

$$\sum_{d \in D} \sum_{\substack{p \in P, \\ e \in p}} z_{dp} h_d \leq c_e \qquad \forall e \in E \tag{12}$$

$$\sum_{e \in p} z_{dp} l_e^* \leq l_d \qquad \forall d \in D, \forall p \in P \tag{13}$$

$$\sum_{p \in P} z_{dp} \geq 1 \qquad \forall d \in D \tag{14}$$

D. PLSCH-MTD: Integrating PLSCH and JSAR

In this section, we propose the integrated model, PLSCH-MTD, to make use of a set of feasible network configurations according to their eligibility together with PLADD schedules. We extend PLSCH to set suitable configurations obtained via JSAR for each scheduled action of the defender. An action represents the replacement of services and re-routing, consuming the limited budget of the defender. At the same time, we enforce a minimum amount of changes between successive configurations. Note that changing the whole configuration may force the attacker to perform a complicated attack once more, but it comes with a certain cost to re-design the network. Eventually, the distance between two configurations deduces a trade-off between reconfiguration overhead and defensive capabilities, e.g., creating a degree of obscurity.

Having a large set of configurations, a defender should decide which configurations can be set after a particular configuration, e.g., which are eligible to be the next configuration. To quantify the eligibility, we propose the following metric, *distance* between two configurations. It is calculated between two configurations c and e as

$$|c - e| = \frac{\sum_{s \in S} \sum_{v \in V} |q_{sv}^c - q_{sv}^e| + \sum_{d \in D} \sum_{p \in P} |z_{dp}^c - z_{dp}^e|}{|S| + |D|} \tag{15}$$

where q_{sv}^c and z_{dp}^c represent the service deployment and demand assignment variables (cf. JSAR) for the configuration c, respectively. The distance between two configurations c

and e is proportional to (i) the number of service migrations, i.e., services migrated to different nodes than the previous configurations, and (ii) reroutings, i.e., traffic streams moved to different paths. Eq. 15 can also be used to calculate the migration overhead that may cause a certain delay and configuration effort for each reconfigured component. We evaluate its effectiveness further in Section VI.

TABLE II: Extended variables and parameters of PLSCH-MTD.

Type	Symbol	Set	Definition
Base	c, e	C	A configuration
Constant	κ	Z^*	Distance threshold
	α_{ce}	Z^*	Indicates if e can be configured after c
Variable	a_{ct}	Z^*	Decides if configuration c set at t

In PLSCH-MTD, we calculate the eligibility of each combination of potential configurations $c \in C$ in advance, considering a threshold distance κ given as input. Two configurations can be set consecutively only if there is a sufficient amount of changes in-between s.t., $|c - e| > \kappa$, which is represented as $\alpha_{ce} = 1$. Table II shows the new parameters and variables introduced with PLSCH-MTD. Accordingly, constraint 16 ensures that any consecutive MTD actions involve two eligible configurations satisfying the given threshold distance.

$$y_{mjt} + y_{mkf} - 1 \leq \sum_{c \in C} \sum_{e \in C} a_{ct} a_{ef} \alpha_{ce}$$
$$\forall m \in M, \forall j, k \in J_m, k = j + 1, \forall t, f \leq T \qquad (16)$$

a_{ct} is a binary decision variable representing if the system is reconfigured with configuration c at time instance t. The quadratic expression in constraint 16 is linearized by using McCormick envelopes [41] to solve the problem easily with state-of-the-art linear optimization tools. Constraint 17 ensures that a respective configuration is assigned at t if there is a defensive action taken s.t. $x_t = 1$.

$$\sum_{c \in C} a_{ct} \leq x_t \qquad \forall t \leq T \qquad (17)$$

Lastly, Constraint 18 avoids the reuse of the same configuration for the given system duration T to prevent an attacker to deduct a reconfiguration pattern.

$$\sum_{t=0}^{T} a_{ct} \leq 1 \qquad \forall c \in C \qquad (18)$$

Note that PLSCH-MTD is an offline solution in which the defender develops a strategy *in advance* against several potential attack scenarios. Therefore, an increasing variety of considered attack scenarios could offer better strategies against broader threats. Besides, it does not require attack detection but can still prevent ongoing attacks by changing the service configuration. It also forces an attacker to rediscover the system with a new configuration. In this sense, it is also a complementary security solution to reactive security mechanisms such as intrusion detection and prevention systems.

V. Attack Scenarios

Several authors of related work tackle single attack scenarios conducted via real security tools. However, they cannot provide an optimal MTD schedule against multiple potential attack patterns [8], [9]. More theoretical related work does not reflect realistic attacks well since they only use probability distribution functions for attack generation [11], [35]. Moreover, data on actual attacks against MCSs is limited to public reports and white papers that partially include attack durations and lack details regarding a complete attack timeline [42]. Although we know rough estimations on the time required for detecting advanced persistent threats [5], [43] and detailed technical analysis of some infamous cyber-attacks and malware [44], [45], it is difficult to obtain the complete picture of specific attack paths and the duration of advanced attacks.

Accordingly, we model different attack types and scenarios considering the recent security incidents in MCSs. An attack scenario is the combination of several individual attack steps as modeled in Section II-B. Those scenarios are then used to evaluate the defensive strategies that PLSCH-MTD provides.

A. Time Characteristics of Individual Attacks

We define three attack types in terms of their duration: long, medium-length, and short attacks. The length of an attack represents its time-to-success value in the PLSCH model. Moreover, we introduce a new variable, Λ, the *attack scale*, to set the relative lengths of different attacks in proportion to a common design parameter. It is defined in a similar scale with the time horizon T (see Section II-B) for a consistent representation of time-related variables. Accordingly, the length of each attack is uniformly sampled from an interval proportional to Λ. The attack types are characterized as follows:

- **Long attack:** It represents the longest phases of an attack scenario, e.g., reconnaissance, developing necessary tools, and executing relatively complicated attack steps. The length of long attacks is sampled from the range of $[0.1\Lambda, 0.3\Lambda]$, s.t., it lasts 20% of a scenario with 10% deviation for $\Lambda = T$.
- **Medium-length attack:** It represents a certain number of successive attack steps that require significant time, e.g., encrypting a large amount of data or doing lateral movement across different network components. A medium-length attack is sampled from the interval $[0.05\Lambda, 0.15\Lambda]$, s.t. it typically takes 10% of a scenario with 5% time deviation.
- **Short attack:** It represents a combination of successive attack steps with short execution time, e.g., changing the configuration of a component, modifying log files, etc. Their length is sampled from the interval $[0.0025\Lambda, 0.075\Lambda]$ taking on average 5% of an attack scenario.

B. Composition of Attack Scenarios

We in the following define four attack scenarios that reflect recent security incidents targeting critical networked systems [4], [44], [46], [47]. They are composed of the attacks described above in dependence on different attacker goals as illustrated in Fig. 4. The duration of an attack scenario, i.e., the total lengths of its individual attack steps, is limited by the time horizon T as it is also considered as the operational time of the system in the PLSCH model.

- **Calibrated attacks:** Calibrated attacks target specific components, technologies, and protocols in an MCS, e.g., although Stuxnet only damages a particular software that operates nuclear centrifuges [4]. Therefore, they require detailed system-specific knowledge and special exploits that induce long reconnaissance and development times. After acquiring access to the system, the attacker conducts a well-targeted sequence of attacks to potentially multiple components. Depending on the target, such attack steps can take different duration to reconnaissance and can be repeated several times [44]. Accordingly, we compose calibrated attack scenarios of (i) an initial *long* attack and then (ii) randomly selected *medium-length* and *short* attacks as many as their total duration stays under T.
- **Lateral movement:** After gaining access to the system, an attacker can move laterally through the network to find critical services or sensitive data. While this still requires an initial reconnaissance time, the attacker should also discover further vulnerabilities to continue its lateral movement [46], which imposes relatively shorter discovery campaigns. Meanwhile, gaining access to the other components potentially requires conducting more spontaneous attacks, e.g., acquiring credentials, patching legitimate software, etc. Accordingly, we compose lateral movement scenarios of (i) an initial *long* attack for reconnaissance, (ii) several *short* attacks for exploitation (between one to three attacks in our model), (iii) *medium-length* discovery periods to move laterally, and (iv) repeating (ii) and (iii) steps through the movement until their total duration reaches to T.
- **Ransomware:** Ransomware attacks spread a generic malware to encrypt files on the target systems and make them inaccessible. These attacks usually start with a phishing attempt, malvertising, or exploiting vulnerabilities in widely-used software [47]. Then, the attacker can wait a long time to discover the most sensitive data or cause the most damage to the target system at the right time. Lastly, it requires several operations for encrypting and copying the respective data. Accordingly, we compose these scenarios of (i) a *medium-length* penetration time using one of the mentioned techniques, (ii) a *long*(er) discovery and activation time, and (iii) *short* operations for obtaining encrypted data as many as their total duration stays under T.
- **Zero-day:** Lastly, zero-day scenarios represent the threats that have not been encountered and thus not analyzed yet. They are composed of randomly-selected *long*, *medium-length*, and *short* attacks with a total duration of T.

Note that Λ only sets the proportion of time-to-success for individual attacks, and its value is not dependent on or limited by T. While higher Λ values provide a higher number of shorter attacks, the opposite results in fewer but longer attacks. This enables us to specify attack scenarios for the desired time duration (depending on T) but still varying timing characteristics (depending on Λ) independently.

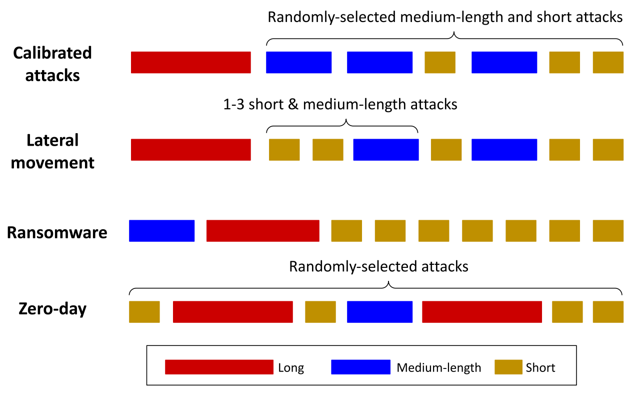

Randomly-selected medium-length and short attacks
Calibrated attacks
1-3 short & medium-length attacks
Lateral movement
Ransomware
Randomly-selected attacks
Zero-day
Long
Medium-length
Short

Fig. 4: Different attack scenarios.

For all scenarios, service reconfiguration within MTD actions helps to invalidate the attacker's knowledge about the system. For instance, service migrations can misorient attackers' movements in lateral movement scenarios. Similarly, against calibrated attacks, service reinitiations can recover the infected services and thus prevent their repetitive malicious behaviors. Finally, in ransomware scenarios, moving the backup data within database services, which is potentially discovered by the attacker, can prevent losing the sensitive data permanently and even disrupt the copying and the encryption processes. These changes also require establishing communication between reconfigured services and the rest of the system, i.e., rerouting data traffic over the network. In this sense, depending on the attack scenarios, the scope of the service reconfigurations can be specified for an MTD strategy.

VI. Evaluation

PLSCH-MTD provides (i) feasible service configurations in terms of resource management and QoS for SOA-based MCSs via JSAR and (ii) optimal MTD schedules on the basis of these configurations to minimize the chance of a successful attack via PLSCH. Accordingly, in this section, we evaluate PLSCH-MTD by answering two main research questions:

RQ1: *How to find a sequence of effective service configurations that utilize the available configuration space efficiently and render the attacker's previous effort obsolete?*
RQ2: *To which extent can an MTD schedule, which is restricted by a limited defender budget, protect the system against several potential attack scenarios?*

In the remainder of this section, we present our evaluation setup, the evaluation metrics, and the experimental results.

A. Evaluation Setup

We implemented our optimization models in CPLEX 12.7.0. All experiments were conducted on a server with 64-core Intel Xeon 2.10GHz CPU and 256GB RAM. We generated random network topologies with $|V| = 20$ and an average connectivity of 1.7, and service overlays with $|D| = 15$ for each experiment as input to the JSAR. The default time horizon T and attack scale Λ values are set to 60. Since we calculated the average number of attacks per scenario as 12 for the $T = \Lambda = 60$, and the time characteristics of attacks (see Section V-A), we set the defender budget to 12 as well, i.e., sufficient budget to prevent all attacks in an ideal scenario. The inter-configuration distance is set to 15% in the joint model, PLSCH-MTD. All other parameter values are given within the respective experiment below. Lastly, we perform 20 iterations per scenario to compute the average results with 95% confidence interval.

B. Evaluation Metrics

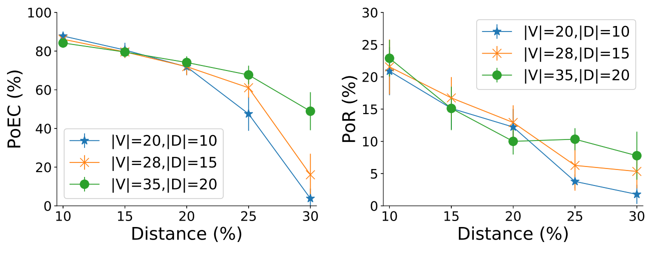
We evaluate JSAR and PLSCH-MTD with different metrics:

- **Percentage of eligible configurations (PoEC):** This is the percentage of configurations that satisfy the minimum amount of required changes between two configurations, i.e., the inter-configuration distance. It indicates how flexible we can use the configuration space for successive MTD reconfigurations.
- **Probability of retain (PoR):** It is the probability that a service instance or a data flow is not migrated after a reconfiguration. It represents whether an attacker can *retain* access to the same service or the data traffic keeping its position, e.g., on the same node or link.
- **Average attacker capture time (ACT):** It is measured by the ratio of the sum of all gaps between consecutive jobs across all machines in PLSCH to the total length of time horizons, i.e., $T * |M|$. The ACT represents the percentage of the total time that the attacker controls the system after a successful attack until the defender takes an MTD action.

While the PoEC and the PoR measures *the effective use of the configuration space* regarding *RQ1*, the average attacker capture time (ACT) measures *the effectiveness of MTD scheduling* to examine *RQ2*.

C. Experimental Results

In this section, we present our numerical results. For our experiments, we use several attack scenarios described in Section V. Multiple instances of a particular scenario type vary due to the randomness in timing characteristics of each attack in a scenario, but still show similar scenario-specific patterns in terms of the order and length distribution of attacks. Then, PLSCH-MTD takes the generated scenarios for each type as input and provides an optimal MTD strategy against them. Alternatively, it is possible to defend against different scenario types at once, e.g., generating several instances per scenario simultaneously, which is referred to as *mixed* scenarios in Section VI. However, note that the length of individual attacks is selected consistently only within a respective scenario type. For instance, while a *long* attack in calibrated attack scenarios can take months, it might be only days in a ransomware scenario. Therefore, it requires selecting a time scale for Λ and T that reasonably models all scenarios.



(a) The impact on the PoEC (b) The impact on the PoR

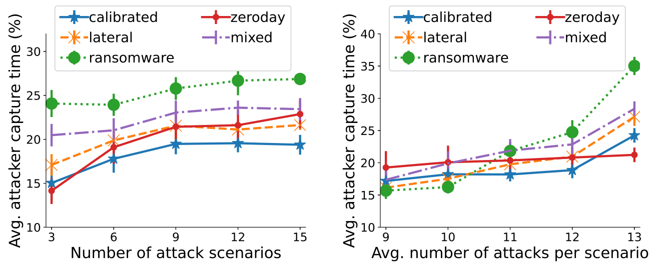
Fig. 5: The impact of inter-configuration distance.

1) *Effective use of the configuration space:*

We first evaluate the effective use of the potential configurations for MTD actions in terms of PoEC and PoR to answer the *RQ1*.

The utilization of configuration space: Fig. 5a shows PoEC for a changing percentage of the minimum inter-configuration distance (κ). The figure contains different graphs for the increasing size of network ($|V|$) and service overlays ($|D|$). An increased minimum distance reduces the PoEC for each network size since it is getting harder to find configurations that are different enough, i.e., with a high inter-configuration distance due to stricter resource utilization. For small networks (blue, star), the percentage converges to nearly 0% at 30% minimum distance requirement. For a larger network (green, dot), in contrast, still around 60% of the potential configurations can be used to reconfigure the service distribution and routing. The results in Fig. 5a indicate that the distance parameter is decisive on (i) having several potential configurations with fewer differences in between or (ii) fewer configurations with more substantial changes. On the one hand, the former enables a defender to utilize distinct configurations for a longer time frame and thus it is harder to detect a reconfiguration pattern for an attacker. On the other hand, a defender should use the same configurations repetitively in the latter scenario, which makes an MTD strategy easier detectable by attackers.

The impact of MTD reconfiguration on attackers: Fig. 5b shows the PoR for an increasing inter-configuration distance and different network sizes. While the 10% distance threshold (κ) gives the attacker on the average a 20-25% chance to access the same service or data that he attacked before the reconfiguration, it converges to nearly 0% for small networks with 30% minimum inter-configuration distance. For larger networks, the PoR is still as low as 10% with a large confidence interval. The reason is, that although the solution space is larger, we do not select particular configurations, e.g., with the maximum distance, but arbitrarily select any two configurations that satisfy the minimum distance requirement. Although an arbitrary selection makes the next configuration less predictable for the attacker, the selection strategy can be adapted, e.g., selecting the configuration with the kth highest distance, to increase his reconnaissance effort. Consequently, a higher inter-configuration distance leads to further changes and enforces the attacker to rediscover the new configuration.



(a) Increasing number of scenarios (b) Increasing number of attacks

Fig. 6: The impact of varying attack scenarios on the ACT.

However, it may also cause service interruptions.

2) Effectiveness of MTD scheduling: We measure the impact of various parameters on the ACT to answer *RQ2*, i.e., how protective an optimal MTD schedule is.

The impact of variety in attack scenarios: We evaluate PLSCH-MTD for each type of attack scenario as well as for *mixed* scenarios that include randomly-selected scenarios simultaneously. Fig. 6a shows the impact of an increasing number of attack scenarios on the ACT. Regardless of the scenario, we observe only a subtle increase from 1% to 3% in ACT. However, defending against multiple scenarios imposes a base challenge that results in 15-25% ACT. The results indicate that although an MTD strategy remains protective against an increasing number of attack scenarios, it is still difficult to defend against even few concurrent scenarios.

The impact of the type of attack scenarios on ACT is more substantial than the impact of their quantity. In Fig. 6a, defending against ransomware is the most challenging with 25% ACT since it consists of several short attacks that can be accomplished. Other scenarios are similarly threatening with 15-18% ACT. Therefore, the effectiveness of PLSCH-MTD is highly dependent on the actual attack scenario.

To evaluate the impact of the number of attacks per scenario, we set $T = 60$ and $\Lambda \in [90, 50]$ (in a reversed order). Decreasing Λ shortens the length of individual attacks and increases their number per scenario, which results in 9-13 attacks for the given range of Λ values. Accordingly, Fig. 6b shows the ACT measurements for increasing attacks per scenario. In the figure, the ACT does not significantly change for 9 to 12 attacks within each attack scenario as there is enough defender budget ($\beta = 12$). This also affirms the results regarding the base challenge (15-25%) of defending against multiple scenarios in Fig. 6a. However, the attacker's success increases by 5-10% for 13 attacks due to the insufficient defender budget.

The impact of defender budget: Fig. 7 shows the impact of an increasing defender budget on the ACT for different numbers of attacks per scenario, i.e., for 18 and 12 attacks by setting $\Lambda = \{60, 90\}$ and $T = 90$. As seen in the figure, more budget strengthens the defender to hold control of the resources with a decreasing ACT regardless of attack counts. When the defender budget is less than the number of attacks per scenario, i.e., for $\beta = 12 - 16$ and $\Lambda = 60$ (solid, blue line), we can observe that a gradual increase in the budget decreases the ACT from 35% to 20%.

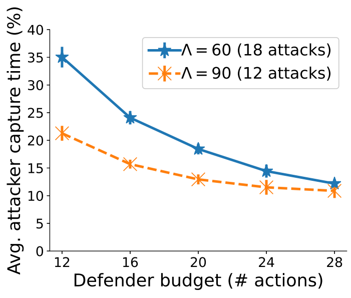


Fig. 7: The impact of defender budget on the ACT.

Theoretically, any defender budget $\beta \geq T$ should guarantee a complete defender occupation. This enables *moving* the system at every possible time instance $t \leq T$ (which is infeasible in practice due to its high overhead) and thus leaves the attacker no chance to accomplish an attack. However, as shown in Fig. 7, it is not possible to obtain that level of protection quickly with a linear increase in the budget after the ACT has converged to 10-12% due to the challenges in defending against multiple attack scenarios.

Note that a single successful attack step may not give the attacker total control over the system as assumed in the PLADD game, but it requires several attack steps to be accomplished. In this sense, our ACT measurements represent *the worst case* that each attack is equally effective. As a result, PLSCH-MTD can still achieve protection of up to 90% of the system operational time with a defensive budget $\beta \ll T$.

VII. Conclusion

Service-oriented architecture (SOA) enables the flexible design of mission-critical systems (MCSs) by dynamically distributing virtual services and establishing their inter-communication. This flexibility can also be utilized to implement moving target defense (MTD) strategies for the security of MCSs. By reconfiguring the critical services and data traffic periodically within MTD strategies, it is possible to protect MCSs against advanced cyber-attacks. In this work, we propose an optimization framework (PLSCH-MTD) by combining a joint service allocation and routing model (JSAR) with an attack-defender game (PLSCH) to find effective MTD strategies for SOA-based MCSs. While PLSCH provides an optimal schedule of subsequent MTD actions against potential threats, JSAR generates feasible service configurations for each action. Furthermore, we model several attack scenarios inspired by the security incidents in MCSs to evaluate PLSCH-MTD. The experiments reveal that the PLSCH-MTD can utilize the service configuration space efficiently to force attackers to rediscover the system. Moreover, it can protect an MCS for up to 90% of its operational time[1].

[1]A preprint of this paper is also submitted to arXiv with the same title.

References

[1] D. Ergenc, J. Rak, and M. Fischer, "Service-Based Resilience for Embedded IoT Networks," in *50th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, pp. 540–551, 2020.

[2] J. Villaneueva, J. Migge, and N. Navet, "QoS-Predictable SOA on TSN: Insights from a Case-Study," in *Automotive Ethernet Congress*, 2021.

[3] T. Cucinotta, A. Mancina, G. F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A Real-Time Service-Oriented Architecture for Industrial Automation," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 3, pp. 267–277, 2009.

[4] T. M. Chen and S. Abu-Nimeh, "Lessons from Stuxnet," *Computer*, vol. 44, no. 4, pp. 91–93, 2011.

[5] M. . FireEye, "M-Trends Special Report," tech. rep., Mandiant & FireEye, 2020. https://content.fireeye.com/m-trends/rpt-m-trends-2020.

[6] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati, "A Survey of Moving Target Defenses for Network Security," *IEEE Communications Surveys and Tutorials*, vol. 22, no. 3, pp. 1909–1941, 2020.

[7] H. Zhang, K. Zheng, X. Wang, S. Luo, and B. Wu, "Efficient Strategy Selection for Moving Target Defense Under Multiple Attacks," *IEEE Access*, vol. 7, pp. 65982–65995, 2019.

[8] M. Thompson, N. Evans, and V. Kisekka, "Multiple OS rotational environment an implemented Moving Target Defense," in *7th International Symposium on Resilient Control Systems (ISRCS)*, 2014.

[9] M. Thompson, M. Mendolla, M. Muggler, and M. Ike, "Dynamic Application Rotation Environment for Moving Target Defense," in *Resilience Week (RWS)*, pp. 17–26, 2016.

[10] M. Van Dijk, A. Juels, A. Oprea, and R. L. Rivest, "FlipIt: The game of stealthy takeover," *Journal of Cryptology*, vol. 26, no. 4, 2013.

[11] S. T. Jones, A. V. Outkin, J. L. Gearhart, J. A. Hobbs, J. D. Siirola, C. A. Phillips, S. J. Verzi, D. Tauritz, S. A. Mulder, and A. B. Naugle, "Evaluating moving target defense with PLADD," tech. rep., Sandia National Lab.(SNL-NM), 2015.

[12] D. Ergenç, J. Rak, and M. Fischer, "Service-Based Resilience via Shared Protection in Mission-Critical Embedded Networks," *IEEE Trans. on Net. and Service Manag.*, vol. 18, no. 3, pp. 2687–2701, 2021.

[13] O. D. Parekh, C. A. Phillips, V. Powers, N. Sakr, and C. Stein, "A Scheduling Problem Motivated by Cybersecurity and Adaptive Machine Learning," tech. rep., Sandia National Lab.(SNL-NM)), 5 2018.

[14] X. Li and C. Qian, "A Survey of Network Function Placement," in *13th IEEE Annual Consumer Communications and Networking Conference (CCNC)*, pp. 948–953, 2016.

[15] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A Comprehensive Survey of Network Function Virtualization," *Computer Networks*, vol. 133, pp. 212–262, 2018.

[16] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual Network Functions Placement and Routing Optimization," in *IEEE Int. Conf. on Cloud Networking, CloudNet*, pp. 171–177, IEEE, 2015.

[17] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating Virtualized Network Functions," *IEEE Trans. on Net. and Service Manag.*, vol. 13, no. 4, pp. 725–739, 2016.

[18] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On Dynamic Service Function Chain Deployment and Readjustment," *IEEE Trans. on Net. and Service Manag.*, vol. 14, no. 3, pp. 543–553, 2017.

[19] J. Chenni Kumaran and M. Aramudhan, "A Survey on Resource Allocation Strategies in Cloud," *International Journal of Reasoning-based Intelligent Systems*, vol. 10, no. 3-4, pp. 328–336, 2018.

[20] N. K. Pandey, S. Chaudhary, and N. K. Joshi, "Resource Allocation Strategies used in Cloud Computing: A Critical Analysis," in *IEEE Conf. on Communication, Ctrl. and Intelligent Syst. (CCIS)*, 2017.

[21] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth, "Modeling and Placement of Cloud Services with Internal Structure," *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, pp. 429–439, 2016.

[22] D. Breitgand, A. Marashini, and J. Tordsson, "Policy-driven service placement optimization in federated clouds," *IBM Research Division, Tech. Rep*, vol. 9, pp. 11–15, 2011.

[23] L. Pu, L. Jiao, X. Chen, L. Wang, Q. Xie, and J. Xu, "Online Resource Allocation, Content Placement and Request Routing for Cost-efficient Edge-caching in Cloud Radio Access Networks," in *IEEE Journal on Selected Areas in Communications*, vol. 36, pp. 1751–1767, 2018.

[24] M. B. Gawali and S. K. Shinde, "Task Scheduling and Resource Allocation in Cloud Computing using a Heuristic Approach," *Journal of Cloud Computing*, vol. 7, no. 1, 2018.

[25] A. Varasteh, B. Madiwalar, A. Van Bemten, W. Kellerer, and C. Mas-Machuca, "Holu: Power-Aware and Delay-Constrained VNF Placement and Chaining," *IEEE TNSM*, vol. 18, no. 2, pp. 1524–1539, 2021.

[26] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing Together the NFV Provisioning Puzzle: Efficient Placement and Chaining of Virtual Network Functions," in *IFIP/IEEE Int. Symp. Integrated Netw. Mgmt. (IM)*, pp. 98–106, 2015.

[27] G. Lee, M. Kim, S. Choo, S. Pack, and Y. Kim, "Optimal Flow Distribution in Service Function Chaining," in *ACM International Conference Proceeding Series*, pp. 17–20, 2015.

[28] L. Askari, M. Tamizi, O. Ayoub, and M. Tornatore, "Protection Strategies for Dynamic VNF Placement and Service Chaining," in *Int. Conf. on Computer Comm. and Networks (ICCCN)*, 2021.

[29] H. Okhravi, A. Comella, E. Robinson, and J. Haines, "Creating a cyber moving target for critical infrastructure applications using platform diversity," *International Journal of Critical Infrastructure Protection*, vol. 5, no. 1, pp. 30–39, 2012.

[30] Y. Huang and A. K. Ghosh, "Introducing diversity and uncertainty to create moving attack surfaces for web services," in *Moving Target Defense*, pp. 131–151, Springer, 2011.

[31] M. Crouse and E. W. Fulp, "A moving target environment for computer configurations using Genetic Algorithms," in *4th Symposium on Configuration Analytics and Automation (SAFECONFIG)*, 2011.

[32] E. Al-Shaer, "Toward network configuration randomization for moving target defense," in *Moving Target Defense*, pp. 153–159, Springer, 2011.

[33] B. Alese, O. Ibidunmoye, D. Haruna, A. Thompson, and I. Otasowie, "Game-based Analysis of the Network Attack-Defense Interaction," *Lecture Notes in Engineering and Computer Science*, vol. 1, 07 2014.

[34] F. Liu, H. Gao, and Z. Wei, "Research on the game of network security attack-defense confrontation through the optimal defense strategy," *Security and Privacy*, vol. 4, no. 1, p. 136, 2021.

[35] S. T. Jones, A. V. Outkin, J. L. Gearhart, J. A. Hobbs, J. D. Siirola, C. A. Phillips, S. J. Verzi, D. Tauritz, S. A. Mulder, and A. B. Naugle, "PLADD: Deterring Attacks on Cyber Systems and Moving Target Defense," in *INFORMS Computing Society Conference*, 1 2017.

[36] Y. C. Chen, V. J. Mooney, and S. Grijalva, "Grid cyber-security strategy in an attacker-defender model," *Cryptography*, vol. 5, no. 2, 2021.

[37] Q. Gao, H. Wu, J. Zhang, Y. Zhang, N. Zhang, and X. Tao, "Multi–Attacker Multi–Defender Interaction in mMTC Networks via Differential Game," in *IEEE/CIC International Conference on Communications in China (ICCC)*, pp. 1250–1255, 2020.

[38] D. Xu, L. Xiao, N. B. Mandayam, and H. V. Poor, "Cumulative prospect theoretic study of a cloud storage defense game against advanced persistent threats," in *IEEE INFOCOM Workshops*, pp. 541–546, 2017.

[39] H. Li, W. Shen, and Z. Zheng, "Spatial-Temporal Moving Target Defense: A Markov Stackelberg Game Model," in *19th Int. Conf. on Autonomous Agents and Multiagent Systems*, p. 717–725, 2020.

[40] S. Sengupta, S. G. Vadlamudi, S. Kambhampati, A. Doupé, Z. Zhao, M. Taguinod, and G.-J. Ahn, "A Game Theoretic Approach to Strategy Generation for Moving Target Defense in Web Applications," in *16th Int. Conf. on Autonomous Agents and Multiagent Systems*, p. 178–186, 2017.

[41] G. P. Mccormick, "Computability of Global Solutions to Factorable Nonconvex Programs: Part I – Convex Underestimating Problems," *Math. Program.*, vol. 10, p. 147–175, Dec. 1976.

[42] P. Pols, "The unified kill chain," *Cyber Security Academy (CSA) Thesis, Hague*, 2017. https://www.unifiedkillchain.com/.

[43] Mandiant, "APT1: Exposing One of China's Cyber Espionage Units," tech. rep., Mandiant, 2021. https://www.mandiant.com/sites/default/files/2021-09/mandiant-apt1-report.pdf.

[44] D. Albright, P. Brannan, and C. Walrond, "Stuxnet malware and natanz," tech. rep., Institute for Science and International Security, 2011.

[45] X. Zhang, O. Upton, N. L. Beebe, and K.-K. R. Choo, "IoT Botnet Forensics: A Comprehensive Digital Forensic Case Study on Mirai Botnet Servers," *Forensic Science International: Digital Investigation*, vol. 32, p. 300926, 2020.

[46] F. Wilkens, S. Haas, D. Kaaser, P. Kling, and M. Fischer, "Towards Efficient Reconstruction of Attacker Lateral Movement," in *14th Int. Conf. on Availability, Reliability and Security (ARES)*, 2019.

[47] M. N. Olaimat, M. Aizaini Maarof, and B. A. S. Al-rimy, "Ransomware anti-analysis and evasion techniques: A survey and research directions," in *3rd International Cyber Resilience Conference (CRC)*, 2021.

Appendix E

SDN-based Self-Configuration for Time-Sensitive IoT Networks

Abstract

The convergence of Information Technology (IT) and Industrial Operations Technology (OT) results in efficient network management solutions for automotive and industrial automation environments. However, configuring real-time Ethernet networks while maintaining the desired QoS is challenging due to the dynamic nature of OT networks and the high number of configuration parameters. This paper introduces a Software-Defined Network (SDN)-based self-configuration framework for the time-sensitive networks (TSNs). Unlike standard TSN, we remove end-host-related dependencies and put streams initially on default paths to extract traffic characteristics by monitoring network traffic at edge switches. Communicated to a central SDN controller, these characteristics allow moving streams to optimal paths while maintaining hard real-time guarantees, for which we also formulate an optimization problem. According to the results, although the proposed approach increases the average delay of critical frames by less than 1%, a certain level of real-time guarantee can be provided without prior knowledge of the streams.

Reference

N. Sertbaş Bülbül, Doğanalp Ergenç, M. Fischer. SDN-based Self-Configuration for Time-Sensitive IoT Networks. IEEE International Conference on Local Computer Networks (LCN), 2021.

Contribution

In the forementioned publication, the contributions of this thesis are modeling and implementing the main optimization model (TSOR) as well as its complexity analysis and integration to the self-configuration framework. The first author designed this framework, implemented a simulation model, and evaluated the overall proposal. She also wrote the majority of the text except the respective section describing the optimization model. The third co-author helped to improve the quality of the paper with his valuable feedback.

SDN-based Self-Configuration for Time-Sensitive IoT Networks

Nurefşan Sertbaş Bülbül , Doğanalp Ergenç, Mathias Fischer
Department of Computer Science, University of Hamburg, Germany
Email:{sertbas, ergenc, mfischer}@informatik.uni-hamburg.de

Abstract—**The convergence of Information Technology (IT) and Industrial Operations Technology (OT) results in efficient network management solutions for automotive and industrial automation environments. However, configuring real-time Ethernet networks while maintaining the desired QoS is challenging due to the dynamic nature of OT networks and the high number of configuration parameters. This paper introduces a Software-Defined Network (SDN)-based self-configuration framework for the time-sensitive networks (TSNs). Unlike standard TSN, we remove end-host-related dependencies and put streams initially on default paths to extract traffic characteristics by monitoring network traffic at edge switches. Communicated to a central SDN controller, these characteristics allow moving streams to optimal paths while maintaining hard real-time guarantees, for which we also formulate an optimization problem. According to the results, although the proposed approach increases the average delay of critical frames by less than 1%, a certain level of real-time guarantee can be provided without prior knowledge of the streams.**

Index Terms—**self-configuration, time-sensitive networks, software defined networking, network management**

I. Introduction

The advent of Industry 4.0 and the Industrial Internet of Things (IIoT) enable new manufacturing scenarios that include advanced robotics, artificial intelligence, smart sensors, and cloud computing. In such scenarios, control of physical processes assumes a time- and safety-critical (and therefore guaranteed) delivery of messages. The IEEE 802.1 working group has proposed time-sensitive networking, TSN, standards to empower regular switched Ethernet with real-time (RT) capabilities. As a result, TSN enables the coexistence of critical time-sensitive and traditional Ethernet traffic with various QoS classes, such as low priority and best effort (BE). It also offers a wide range of functions for RT systems, such as time synchronization, reliability, scheduling, and network management.

In TSN, the management and configuration of a network are described in the IEEE 802.1Qcc stream reservation protocol (SRP) standard [3]. SRP specifies how to schedule a time-sensitive stream by allocating the required network resources. Moreover, the standard defines alternative network configuration and management schemes that leverage SRP. Several studies are suggesting that complementing the TSN with a networking concept such as software-defined networks, SDN, is a beneficial configuration solution [4], [5], [18]. With additional protocols (e.g., Netconf and Openflow), SDN allows for instant configuration of routes and transport schedules based on a central control plane [10]. It also allows split up flows for transmission on multiple paths for load-balancing, using the available bandwidth more efficiently, and making network-wide configurations such as time-synchronization.

However, the proposed configuration schemes rely on the active participation of end hosts to communicate service features and communication requirements to a centralized or decentralized management component. This approach requires the manual configuration of highly heterogeneous edge hosts to demand the necessary resources from the network. It can be edge hosts, low-power sensors and actuators, entire cyber-physical systems, or robots that may or may not support the required TSN registration protocols. For large systems and many connected end hosts, even with SDN, their configuration can be cumbersome and requires ongoing maintenance. Therefore, we believe that plug-and-play self-configuration can help adopt existing TSN protocols for future networks and devices. However, the self-configuration of TSN networks is not part of the current standards.

The main contribution of this paper is a novel SDN-based self-configuration approach for TSN networks in IoT scenarios. In our approach, end-hosts do not need to be TSN-aware, and they obtain the required network resources transparently. With that, we eliminate the talker responsibility of propagating new traffic parameters each time. That eases the configuration specifically for highly dynamic environments with a large number of hosts. Accordingly, our contributions are:

- We introduce a self-configuration approach on the basis of SDN for TSN networks and this at the expense of marginal additional delay for the routing of streams.
- We formulate the time-sensitive optimal routing (TSOR) model as a mixed-integer linear programming (MILP) model. TSOR considers the optimal routing problem together with the service-based stream configuration regarding the main characteristics of TSN.
- We propose a learning component that detects traffic characteristics and eases the SRP process for various scenarios.
- We evaluate our approach via realistic OMNet++ simulations. Our evaluation results indicate that we can extract related traffic parameters in near real-time. That results in a slight increase in end-to-end delay only for less than 1% of time-triggered (TT) traffic.

The remainder of this paper is structured as follows: Section

II summarizes related work on TSN stream registration. Section III describes current TSN configuration approaches. In Section IV, we introduce our overall architecture. We evaluate our approach and describe our simulation results in Section V. Finally, Section VI concludes the paper and summarizes future work.

II. Related Work

In this section, we present the literature survey on the configuration of TSNs. Offline scheduling approaches as in [13] statically allocate network resources for the given communication patterns, e.g., TT traffic. That approach works in specific scenarios, e.g., automotive systems, where the communication streams are already known at design time. However, to meet the high priority QoS requirements of future industrial networks, dynamically routing packets depending on the current state, e.g., switch workloads, requires a dynamic configuration, including a dynamic resource allocation.

For TSN, a configuration of the network resources to transfer TT traffic is described in IEEE 802.1Qcc [3] on the architectural level. Due to lack of concrete specification, the authors of [6] propose a configuration architecture named Software-Defined Flow Reservation based on OpenFlow protocol. However, they only describe the essential components as proof of concept to manage network resources in RT and register time-sensitive streams while routing and scheduling mechanisms are left out of scope. In [12], a generic concept for secure and time-sensitive communication in industrial networks is described. Similar to [6], there is no further evaluation or the details of implementation. Besides, the configuration of the RT traffic is left as an open issue.

In [8], a stream-specific bandwidth and buffer capacity reservation mechanism is proposed. Global knowledge of the controller is used in routing to compute an appropriate network configuration. They also simplify the end-hosts by removing clock synchronization and employ a time-division multiple access mechanism. However, their MILP-based path-finding approach is too complex to deliver results in RT. In [16], a combined routing and scheduling algorithm is proposed for incrementally adding or removing time-sensitive streams at runtime. The approach schedules transmission at the edges, which requires only limited schedule updates. While it does not require any configuration on switches, it assumes that hosts have proper clock synchronization and are involved in the scheduling process.

These studies mainly focus on TT traffic under significant assumptions such as having apriori information about the traffic and TSN-aware clock synchronized hosts. Authors in [9] propose a concept of a configuration agent including a monitor, an extractor, and a scheduler component to make RT switches self-configurable. However, they consider only TT traffic and left sporadic traffic as future work. Also, they propose an abstract end-to-end architecture and do not evaluate the overall system.

III. Background on IEEE 802.1Qcc

In TSN, the configuration starts at end-hosts named talkers and listeners, the source and destination nodes. A talker sends its specific traffic requirements to the edge switch to request network resources and scheduling. Then, this switch either (i) computes the required resources and schedules for the related traffic and forwards the request to other switches or (ii) directly forwards the request to a central controller that can configure all the switches on the path towards the listener accordingly. Afterward, the talker starts sending frames to the network.

In the rest of this section, the background information on the TSN configuration models is given, including the description of the models, user configuration parameters, and the stream reservation protocol.

A. *TSN Configuration Models*

In the current standard, three configuration schemes are described at the architectural level.

In the **fully distributed model**, an end-host communicate with the edge switch to declare its traffic requirements, and the switch forwards the requirements to the other core switches in the network (See Fig. 1-a). Here, switches are not configured by a central entity but in a distributed manner with their local knowledge. Such a configuration is not suitable for mechanisms that require collaboration between bridges, e.g., scheduling via time-aware shapers [2].

In the **centralized network/distributed user model**, user configuration is still distributed, and edge switches share requirements of the end-hosts with a central entity named central network configuration (CNC) instead of propagating them through other switches (see Fig. 1-b). Since some scenarios, such as gate configuration at the switches, require network-wide knowledge and high computational power, CNC offers

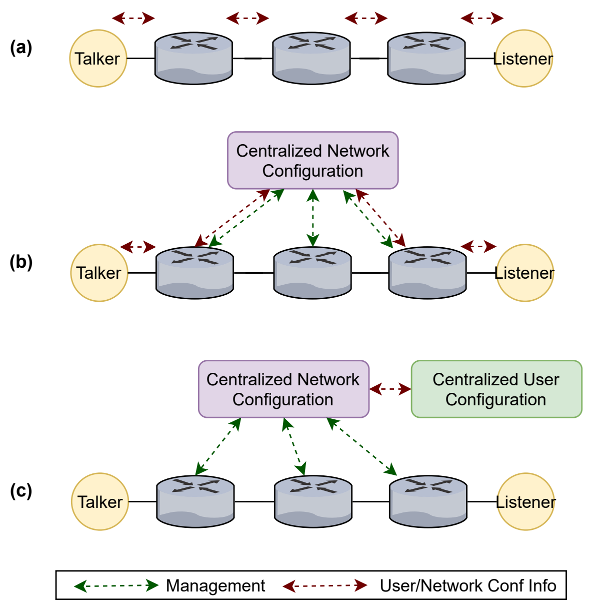

(a)
Talker
Listener
Centralized Network Configuration
(b)
Talker
Listener
Centralized Network Configuration
Centralized User Configuration
(c)
Talker
Listener
Management
User/Network Conf Info

Fig. 1: TSN configuration models.

a better configuration with its global knowledge and higher computational capabilities than forwarding plane elements.

In the **fully centralized model**, both user and network configurations are centralized by centralized user configuration (CUC) and CNC (see Fig. 1-c). End-hosts communicate directly with the CUC for requirement declaration. Unlike the previous models, the CUC configures the end-hosts, and this also involves further interaction with the end-hosts. In this way, packet transmission schedules of the end-hosts are configured, which might be required to satisfy strict timing requirements.

Regardless of the model, there are two types of configuration information exchanged between end-hosts and the network; talker/listener request and status as a reply. The talker/listener request includes several fields such as transmission parameters (e.g., max frame size and frame interarrival time) and stream identifier. A reply message contains status information such as related StreamID, the status of the current stream configuration, and failure information if a failure exists.

B. Stream Reservation Protocol

SRP is an extension of the IEEE 802.1Q standard that describes how to manage resource reservations in LANs [3]. It defines how to specify and propagate talker registrations through the network with guaranteed QoS. SRP runs at bridges by recording relevant information about the connected end-hosts, such as communication latency between a talker and a listener and current stream registrations. The bridges use such information to provide guaranteed QoS for the TSN streams.

SRP can be used in a centralized and a distributed manner as defined in [3]. A distributed model only helps to configure a limited number of parameters with the local information in a switch. In the centralized model, SRP can be used to communicate between the talker/listener and CNC. Initially, the talker requests the required bandwidth resources for a stream. As long as there are sufficient bandwidth resources on a selected path for the stream, that capacity is allocated for the related stream, and the switches are configured accordingly. SRP also enables talkers/listeners to join later or leave. However, it requires direct messaging between the end-hosts and the switches.

As mentioned, SRP requires the active involvement of the end-hosts through that resource reservation process. Here, our goal is to remove such end-host-related dependencies. Accordingly, in the next section, we present our TSN self-configuration approach in detail.

IV. TSN Self-Configuration Approach

In this section, we introduce our SDN-based dynamic self-configuration approach for TSN that we name SC-TSN. In SC-TSN, we remove end-host-related dependencies of standard TSN in which hosts need to actively communicate their traffic requirements. Instead in SC-TSN, edge-switches automatically learn traffic characteristics by routing streams via default paths first and then migrating them once the characteristic is known.

In the remainder of this section, we first describe the overall framework, then we explain how we extract traffic characteristics, and how we compute paths for TSN streams.

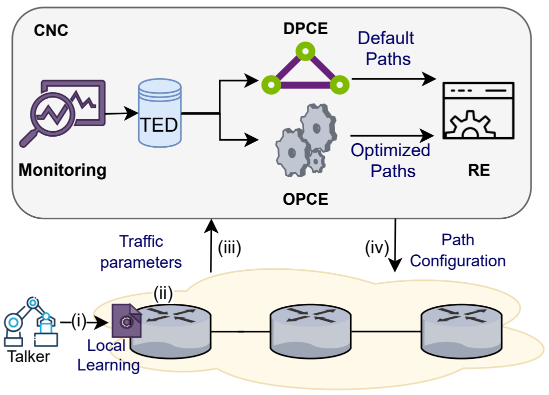


Fig. 2: Overall system architecture.

A. SC-TSN Overall System

We follow the distributed user and centralized network configuration model for our system design as shown in Fig. 2. In contrast to standard TSN, end-hosts directly start communicating via the edge switch (i) and the edge switch extracts traffic characteristics seamlessly (ii). Here, switch treats the traffic like low priority traffic until the traffic characteristics are extracted. Then, the extracted characteristics are forwarded to the SDN-enhanced CNC (iii). The global network view of the CNC enables highly optimized flow assignments and a fast response to varying demands. For that, the CNC computes paths and installs the required flow rules (iv).

All streams are initially perceived as BE traffic unless otherwise is declared, e.g., pre-configuration might still be necessary for safety-critical applications. Then, these streams are forwarded via the default paths without resource reservation until we have successfully obtained their characteristics (see Section IV-B). A default path is defined as a path with sufficient link capacity for immediate and temporary use but not necessarily optimal. With that, we decrease the configuration delay until an optimal path is being found. Such paths are computed in the background by the *Default Path Computation Element (DPCE)* (see Section IV-C). The required information for computing paths such as network topology and current network status, e.g., link utilization, is obtained via the *Monitoring Module*. It collects OpenFlow statistics from the data plane and stores them in the *Traffic Engineering Database (TED)*. Then, DPCE uses this information to compute paths based on the current network status.

In the meantime, the edge switch analyzes the received streams to learn their traffic characteristics and derive their resource and scheduling requirements. For that, we empower edge switches with learning capabilities to extract the traffic patterns such as the frame period p and the maximum frame interarrival time p_{max}. Suppose the stream is classified as TT after a certain time. In that case, the *Optimal Path Computation Element (OPCE)* computes an optimal path for that stream on the fly by solving the optimization model TSOR (see Section IV-D). Then, the stream is migrated to the new path via the *reconfiguration element(RE)*. We also monitor streams in the aftermath to ensure that they still transmit with the extracted traffic parameters. When the characteristics of a stream change, we calculate the deviation from the previously extracted period, restart the learning procedure, and update the configuration, which might induce another stream migration.

Note that SC-TSN does not intend to replace the existing SRP mechanism completely. Instead, it is a hybrid mechanism compatible with the current standards. Even though SC-TSN does not presume information about stream characteristics, we might still use an SRP-like stream registration to declare end-host requirements directly. Since switches support 802.1Q priority levels, such a configuration can be used to ensure a certain level of service guarantee for highly critical applications. SC-TSN is helpful for less critical application scenarios that generate sporadic traffic, e.g., BE or event-triggered (ET), which starts at an arbitrary time. Even though abruptly changing traffic patterns in critical systems is not very common, hosts can change their traffic behavior during runtime. With SC-TSN, we could directly handle such changes dynamically without waiting for further end-host declarations. Thus, SC-TSN helps to configure small to large-scale systems where different traffic types such as cyclic/periodic (e.g., signal transmission) or acyclic/sporadic (e.g., event-driven) can coexist.

B. Learning Traffic Parameters

As explained previously and according to the TSN standards, the talker informs the network controller about its traffic requirements before the actual communication starts. That requirement specification includes frame size and interarrival time of the frames, which are used to allocate the required resources. In contrast, in SC-TSN the edge switches learn traffic parameters by observing the traffic at the network's ingress. These edge switches are enhanced by learning capabilities to analyze receiving traffic to extract related parameters. Since we try to learn traffic characteristics at the edge, we do not need to consider interference from other traffic as in switch-to-switch links. However, we still need an intelligent solution here instead of getting the average interarrival time as a period.

In the signal processing literature [7], [17] analyzing sequences in the frequency domain with Fourier transformation and autocorrelation for periodicity detection is widely used. The Fourier transformation works well for short periods, but may generate many false positives. Thus, the authors of [19] combine Fourier transformation and autocorrelation to detect both short and long periods. In this paper, we use this approach for learning the necessary TSN stream characteristics.

We record the arrival time of the frames for each stream and then try to find the period in the frequency domain. For that, we first transform observations to a time sequence $x_t = x_{t_1}, x_{t_2}, ...x_{t_n}$ where $x_{t_k} = 1$ means that a frame arrived at t_k. Then, we look at the signal power spectral density by computing the discrete Fourier transform to identify the frequencies that carry most of the energy. In other words, the power spectral density analysis can discover the most dominant periods. These periods are then validated with

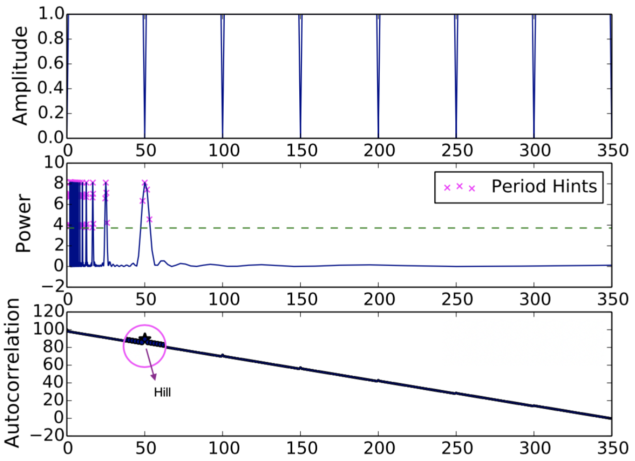


Fig. 3: Period extraction for a sample sequence.

autocorrelation. In that phase, if the candidate period stays at the valley of the autocorrelation function, it is interpreted as a false alarm and is discarded. Otherwise, it is considered a valid period. The period extraction steps are illustrated in Fig. 3. As can be seen from the power spectrum, there are several period candidates that need to be further analyzed by autocorrelation. The periods that stay at the hill (as seen in the autocorrelation plot) are verified as an exact period. When the *Learning Module* detects this period, it triggers the *OPCE* to compute the optimal paths for the extracted parameters.

C. Default Path Computation

To compute default paths for low priority streams, we use a link-utilization-based shortest path algorithm. Furthermore, we use dynamic link weights that the SDN controller updates based on the current link utilization.

To increase the stability of the forwarding tables and limit path changes, we follow the methodology proposed by [20]. We summarize this methodology in pseudo code in Algorithm 1. For each link, we map the current link utilization, u_i, to the link weight, W_i^n, via a linear weight mapping function f. Due to the used mapping function, the link weights remain fixed for low utilization values, which keeps the routing overhead low. Then, we compute the weighted average of the last three-link weights, W_i^{new}. We only update the link weight if the change exceeds the threshold, e.g., $\theta = 20\%$ of its

Algorithm 1: Link Weight Update Process

```
Current link utilizations $\mathcal{U} \leftarrow [u_1, u_2, u_3, ...u_k]$
foreach $u_i \in \mathcal{U}$ do
    $W_i^n = f(u_i)$
    $W_i^{new} = \alpha_1 W_i^n + \alpha_2 W_i^{n-1} + \alpha_3 W_i^{n-2}$
    if $(W_i^{new} - W_i^{n-1}) \geq \theta$ then
        $W_i^n \leftarrow$ set to $W_i^{new}$
    else
        $W_i^n \leftarrow$ set to $W_i^{n-1}$
```

previous value. Finally, we compute the shortest paths with the updated link weights.

With that, *DPCE* can dynamically update link weights and computes new paths with the shortest path algorithm. In case of path changes, it will send new flow rules to update the flow tables of the related switches. Then, these paths are stored to be used for low-priority streams.

D. Optimal Path Computation

By utilizing the *Learning Module* and *DPCE*, our system can extract the traffic characteristics of an incoming stream at edge switches. Depending on the link utilization, it selects the default paths to deploy low priority streams, which does not always require to assign them to their optimal paths.

However, high-priority streams with strict timing requirements cannot be assigned to the default paths as it might result in missing deadlines. Once a stream is classified to have a high priority, its extracted parameters are passed to the *OPCE* to compute the optimal path. Accordingly, we formulate TSOR as a MILP model to be used by *OPCE* as an optimization framework to migrate high-priority streams to suitable paths regarding their time-sensitive requirements.

Using the model, we find (i) end-to-end paths for given demands under different QoS requirements within limited network resources and (ii) gate configurations for each switch that minimizes the overall end-to-end communication latency. The gate configuration is the primary mechanism of the core TSN protocol, IEEE 802.1Qbv Time-aware Shaper (TAS) that ensures end-to-end deterministic communication via strict time-division scheduling for the streams of different QoS classes [2], [15]. In TAS, on each (egress) port of a switch, there are eight priority queues that store frames of streams with different priorities, including best-effort, before they are forwarded to the destination. Each queue is controlled by a gate to forward a frame. When a gate is open, the next frame in the respective queue is sent at a given time. Eight gates corresponding to the eight priority classes are configured by a gate driver via a gate control list (GCL) that decides which gate(s) should be open at which time. This mechanism overall constitutes a frame-forwarding schedule with respect to the priority classes to satisfy strict timing requirements. Eventually, the gate configuration is the prominent feature of the optimization model that enables to derive port-based flow assignment regarding capacity and delay requirements and combines the routing problem with the characteristics of TSN.

In TSOR, we utilize two optimization variables. x_{dp} is a binary variable to decide if demand $d \in D$ is assigned to directed path $p \in P_d$. Here, each d is defined between a talker and a listener, where D is the set enumerating all demands. Accordingly, P_d represents the set of paths computed between those two particular end-points. g_{es}, is a continuous variable defined within $[0, 1]$ and represents the frequency of an open gate on the egress port of link $e \in E$ for the service class s. Thus, g_{es} specifies the priority given to service class s on a directed link e. While $g_{es} = 1$ infers that the gate for s should be open all the time and the capacity of the entire link e is used for that type of demands, $g_{es} \approx 0$ means that any demand of service type s is not active at all on the respective port and thus, the gate is closed. Otherwise, the respective gate for the service class s on link e is open as proportional to $0 < g_{es} < 1$. From this perspective, g_{es} is affected by the total required resources for the demands of service type s as the available capacity, e.g., bandwidth, of e is distributed among those demands according to their service type. Note that each demand is associated with a service class according to the evaluation of the *Learning Module*.

The constraints and the objective function of TSOR are described below.

$$\sum_{p \in P_d} x_{dp} = 1 \qquad \forall d \in D \qquad (1)$$

Constraint (1) is defined to ensure that each demand $d \in D$ is assigned to exactly one path $p \in P_d$. Note that we assume here that all flows are non-bifurcated, e.g., not divided into multiple paths.

$$\sum_{d \in D} \sum_{p \in P_d} x_{dp} \alpha_{ep} h_d \leq c_e \qquad \forall e \in E \qquad (2)$$

Constraint (2) is the link capacity constraint and guarantees that each link e has sufficient capacity c_e to handle the total load h_d of all demands $d \in D$ assigned to any path p having e, s.t. $\alpha_{ep} = 1$.

$$\sum_{s \in S} g_{es} = 1 \qquad \forall e \in E \qquad (3)$$

Constraint (3) represents the configuration of the gate control list of e for each class of service s. As the gates, i.e., enabling queues of an egress port, share limited link resources, only a set of them can be practically open at the same time. Here, a gate for class s is decided to be open on link e as proportional to the value of g_{es}.

$$\sum_{p \in P_d} \sum_{e \in E} x_{dp} \alpha_{ep} \left[l_e^o + l_e^q (1 - g_{es}) \right] \leq l_d \qquad \forall d \in D \qquad (4)$$

Constraint (4) is the latency constraint to ensure that the end-to-end latency on path p is always below the latency requirement of demand d, which is l_d. Considering that s is the service class of d, the gate configurations g_{es} for that service class through the all link e belongs to path p, s.t. $\alpha_{ep} = 1$, impacts the end-to-end latency. Note that while higher values of g_{es} positively impact the latency at link e as it enables the traffic of service type s more often, smaller g_{es} causes an increased latency due to queueing delay in the respective gate. Accordingly, we add the delay factor l_e^q to proportion $1 - g_{es}$ and to represent the queueing delay. Apart from that, a base delay l_e^o representing the port and link characteristics, e.g., packet processing and propagation delay, is considered for each link. While those design parameters, l_e^q and l_e^o, can be set according the system and network properties, we use

$l_e^q = 0.5$ and $l_e^o = 1.0$ in our simulations.

$$g_{es} - \sum_{d \in D} \sum_{p \in P_d} x_{dp} \alpha_{ep} \frac{h_d}{c_e} \geq 0 \qquad \forall e \in E, \forall s \in S \tag{5}$$

Constraint (5) forces g_{es} to be proportional to the total traffic load of service type s forwarded through the link e. Otherwise, it would lead to packet drops due to the congestion.

$$x_{dp} \geq a_{dp} \qquad \forall d \in D, \forall p \in P_d \tag{6}$$

Lastly, constraint (6) fixes the demands that are already assigned to a certain path p, i.e., $a_{dp} = 1$. Using a_{dp}, TSOR can assign incoming demands incrementally without violating a potential set of already-assigned demands. a_{dp} is given as input to the problem. Note that although keeping the previous demands fixed before allocating a new demand reduces the flexibility of routing, it is essential to have a stable configuration scheme, especially for the critical and high-priority demands. That is, reconfiguring the network also has a certain cost, e.g., delay for migrating demands, sending control packets to the switches, and can hinder the deterministic communication requirements. The evaluation of that cost might be critical for real deployments, but it is out of the scope of this paper.

$$\min \sum_{d \in D} \sum_{p \in P_d} \sum_{e \in E} x_{dp} \alpha_{ep} \left[l_e^o + l_e^q (1 - g_{es}) \right] \tag{7}$$

Our objective function (7) minimizes the overall latency of the selected paths, which is calculated similar to the latency constraint (4).

Regarding the complexity, TSOR has $\mathcal{O}(|D||P| + |E|)$ optimization variables where the number of paths is proportional to the total number of links $|E|$. Note that even though g_{es} depends on the number of service classes $|S|$, it is fixed to eight at the TSN context and thus, we assume that as a constant. In terms of the number of constraints, TSOR is bounded by $\mathcal{O}(|D||P| + |E|)$ constraints with the same assumption on the number of services.

Another important complexity issue of TSOR is the linearization of non-linear constraints and the objective function. The model with all linear equations makes the problem more convenient to be solved by state-of-the-art linear optimization tools. Therefore, we linearized the multiplication of a binary variable x_{dp} and non-binary variable g_{es} in the constraint (4) using McCormick envelopes [14] introducing some additional complexity.

V. Evaluation

This section evaluates SC-TSN and compares it to a TSN configuration with SRP for varying traffic load and topology sizes. First, we explain the evaluation setup and metrics. Then, we summarize our evaluation results.

A. Experimental Setup

We implemented SC-TSN in OMNeT++ v5.5.1 using its INET framework and extending the SDN4CoRE framework [10]. SDN4CoRE enables to configure both SDN and TSN capable switches. We developed four applications: *OPCE*, *DPCE*, *Monitoring*, and the switch learning module. To find the optimal path assignment for streams, we implemented the TSOR presented in Section IV-D in CPLEX 12.7.0.

In our experiments, we used three real-world network topologies from the Topology Zoo dataset: *Getnet*, *Integra*, and *Garr201001* as summarized in Table I [11]. A given topology node is mapped to an edge switch with learning capabilities if its node degree is smaller than the average node degree and as a backbone switch otherwise. We assumed that end-hosts are connected only to edge switches.

Since different service classes (e.g., TT and BE) can coexist in the same TSN network, we generated mixed traffic scenarios for a comprehensive evaluation. For TT traffic, we uniformly selected talker-listener pairs whose packet sending periods are chosen uniformly between 2-20 ms as stated in [1]. We initiated the TT traffic at different time instances and set the fixed frame size as 1522 bytes as in [10]. For BE traffic, we use the same packet size (i.e., 1522 byte) and exponentially distributed packet interarrival times [15]. We set the same packet generation rate at each BE traffic source and configured them to start transmission at the beginning of the simulation. We also set our simulation time to 50 seconds and statistic collection period to 2 seconds for link weight updates. The results are given with a 95% confidence interval.

We compare our approach with SRP, which is explained in Section III-B. In SRP, everything is given, and all the conditions for an optimal deployment are already there before the actual communication starts. Thus, it is the ultimate competent for SC-TSN. Since SC-TSN needs sufficient time to extract traffic behavior with confidence, we want to show that frames handled during that phase will not suffer from significantly increased latencies. For that, we use the following metrics:

- **End-to-end (E2E) latency**: The latency of frames until they reach their destination.
- **Delayed TT frame rate**: The ratio of TT frames whose E2E latency is larger than its period to the total frames.
- **Classification rate (CR)**: The ratio of correctly classified TT and BE frames to the total frames.
- **The true negative rate (TNR)**: The ratio of correctly classified BE frames.

TABLE I: Topologies used in simulation.

Metrics \ Networks	Getnet	Integra	Garr201001
Average node degree	2.29	2.67	2.52
# of edge switches	4	16	38
# of backbone switches	3	11	16
# of edges between switches	8	36	68
# of hosts per switches	10	5	2
Total number of nodes	47	107	130

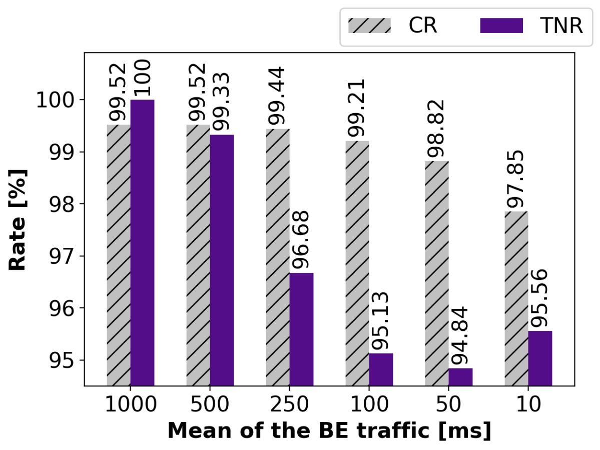


Fig. 4: Classification performance of the learning module.

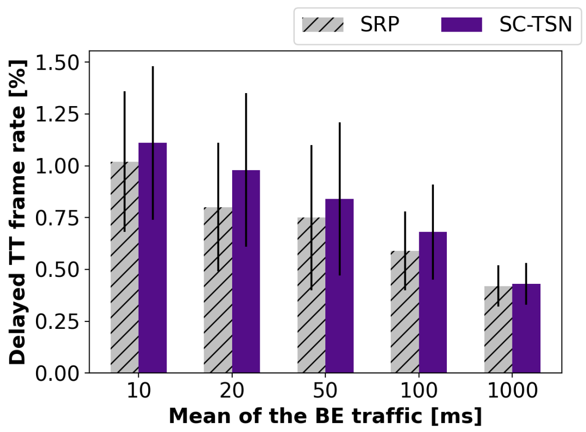


Fig. 5: Delayed TT frame rate.

B. Results

In this section, we first summarize our results on the performance of the learning module at edge switches. Then, we compare SC-TSN with the SRP-based configuration in terms of the end-to-end latency.

1) Performance of learning traffic parameters: We evaluated the classification performance of our learning module for BE and TT traffic under different BE loads. For that, we kept the same TT streams for each experiment and varied the interarrival time of BE frames, and measured CR and TNR. Fig. 4 shows the accuracy independent of the interarrival time of BE frames. The results indicate that we can classify in between 99.52% and 99.85% of TT and BE frames correctly. We also see that the TNR is between and 94.84% and 99.52% for different rates of BE traffic. For light BE load, e.g., when the mean of BE traffic is 1000ms, we can classify almost all BE streams correctly. However, when the interarrival time decreases, our learning approach starts to classify BE frames as TT. We run these experiments directly in the simulation environment because even though we use the same traffic loads, different factors such as queuing delays affect the frames' arrival time.

To sum up, our results indicate that the BE classification rate does not change significantly with an increasing load. In case of the misclassification of BE traffic as a TT, we use the optimal paths instead of the default ones. This may decrease the E2E latency of BE frames. However, the misclassification of TT traffic around 0.5% does not significantly affect E2E TT latency because only the first few frames of each TT stream are misclassified. In that case, only those frames are sent via the preconfigured default paths.

2) Impact of learning on the delivery performance: We measure how the delay of TT streams is affected by an increasing BE traffic. We used the Integra topology and set the number of TT streams to half of the number of nodes and BE streams to half of the number of the TT streams. Then, we repeated the experiment for different interarrival times (μ) of the BE frames, from 10ms to 1000ms as in Table II. We measured the latency and the delayed frame rate. As expected, SRP and SC-TSN are quite close; they have nearly the same average and minimum TT latency values. Since we use stream priorities at the switches, the average latency of the TT frames is not significantly affected by the increasing load of the BE traffic. However, we observe an increase in the maximum latency. Our approach has a higher maximum latency than SRP because of the learning process. Before the extraction of the exact period, the received frames are routed as low priority traffic; if otherwise is not preconfigured, and send via the default routes. Therefore, they might be significantly delayed. To check this, we measure the delayed TT frame rate, as seen in Fig. 5 and we observe that SC-TSN has a higher delayed frame rate as expected.

Our learning module may classify BE frames as high priority and send over optimal paths as explained previously. We see that SC-TSN has lower BE latency than the SRP between 10 ms to 50ms. Even though it seems like the BE classification rate increases in that interval (see Fig. 4), the number of BE frames is also increasing. In contrast, the number of TT frames remains the same. Thus, the effect of misclassified BE frames becomes more visible and we observe lower BE latency in SC-TSN, as shown in Table III.

Lastly, we measure how TT frames are affected by network

TABLE II: E2E latency of TT frames for varying BE load.

	SRP			SC-TSN	
Mean BE traffic	**Mean [ms]**	**Max [ms]**		**Mean [ms]**	**Max [ms]**
10ms	1.31	10.52		1.35	17.30
20ms	1.30	4.31		1.32	11.44
50ms	1.29	2.67		1.30	8.08
100ms	1.29	2.48		1.30	6.24
1000ms	1.29	2.47		1.29	5.80

TABLE III: E2E latency of BE frames for varying BE load.

	SRP			SC-TSN	
Mean BE traffic	**Mean [ms]**	**Max [ms]**		**Mean [ms]**	**Max [ms]**
10ms	1.57	119.85		1.54	121.24
20ms	1.56	121.2		1.55	121.50
50ms	1.45	115		1.45	118.28
100ms	1.38	102.1		1.36	101.1
1000ms	1.38	73.77		1.36	72.57

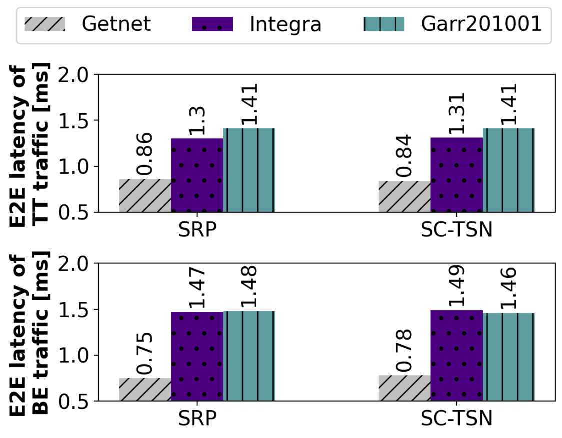


Fig. 6: Performance comparison for varied sized topologies.

topologies of different sizes as given in Table I. As in the previous experiments, we set the number of TT streams proportional to the number of nodes in the network. Thus, we have 23 TT and 11 BE sources in Getnet, 53 TT and 26 BE sources in Integra, and 65 TT and 32 BE sources in Garr201001 topologies. Fig. 6 shows that the time for solving the optimization problem does not significantly affect the E2E latency for small topologies such as Getnet. However, for medium- and large-size topologies, Integra and Garr201001 in our setup, we observe that the latency increases quickly. A critical finding at that experiment is that the latency of TT and BE frames converges in the larger topologies since the solution time of the optimization problem increases with the topology size.

VI. Conclusion

Configuration of TSN is a challenging task and requires considerable engineering efforts. Although the alternative configuration schemes have been introduced in the IEEE 802.1Qcc standard, the self-configuration of TSN is not covered. This paper proposes an SDN-based hybrid self-configuration framework for the TSN networks, SC-TSN, in accordance with the plug-n-play nature of Ethernet networks. In that sense, end-hosts do not need to declare their traffic requirements in advance. Instead, the SC-TSN adapts itself to the stream traffic requirements and reserve the required resources for routing the data traffic. SC-TSN also allows for an SRP-like stream registration procedure via the SDN Northbound API for highly critical traffic. Experiments indicate that SC-TSN can successfully detect traffic characteristics with an over 97.85% classification rate. Moreover, it does achieve results close to SRP with a minimal increase in the E2E latency and the delayed frame rate.

As explained in [2], bounded latency for TT frames can be assured by configuring which 802.1Q priorities are allowed to pass through a particular port at a specific time. Otherwise, E2E latencies are negatively affected by each traversed switches' queuing delays on the multi-hop routes. Since we do not use time-aware gates at switches, it is challenging to guarantee bounded latency. However, the configuration of gate control lists is possible with the SDN, as shown in [10]. As we consider the gate configuration in our optimization model, TSOR, it is also a valuable future work to extend our whole design, including gate-configuration features.

As a part of future work, we plan to examine the performance of SC-TSN with more comprehensive scenarios where the traffic patterns are dynamically change and eventually triggers re-routings much more often. Also, evaluating SC-TSN in network failure scenarios requiring sudden route changes could be another possible study.

References

[1] "Time sensitive networks for flexible manufacturing testbed - description of converged traffic types." [Online]. Available: https://www.google.com/url?sa=t%2C

[2] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," pp. 1–57, 2016.

[3] "IEEE standard for local and metropolitan area networks–bridges and bridged networks - amendment 31: Stream reservation protocol enhancements and performance improvements," 2018.

[4] J. L. Du and M. Herlich, "Software-defined networking for real-time ethernet," *ICINCO*, 2016.

[5] M. Ehrlich, D. Krummacker, C. Fischer, R. Guillaume, S. S. P. Olaya, A. Frimpong, H. de Meer, M. Wollschlaeger, H. D. Schotten, and J. Jasperneite, "Software-defined networking as an enabler for future industrial network management," in *ETFA*. IEEE, 2018.

[6] T. Gerhard, T. Kobzan, I. Blöcher, and M. Hendel, "Software-defined flow reservation: Configuring ieee 802.1 q time-sensitive networks by the use of software-defined networking," in *ETFA*. IEEE, 2019.

[7] R. Gove and L. Deason, "Visualizing automatically detected periodic network activity," in *VIS*. IEEE, 2018.

[8] J. W. Guck and W. Kellerer, "Achieving end-to-end real-time QoS with software defined networking," in *CloudNet*. IEEE, 2014.

[9] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, "A configuration agent based on the time-triggered paradigm for real-time networks," in *IEEE WFCS*, 2015.

[10] T. Häckel, P. Meyer, F. Korf, and T. Schmidt, "SDN4CoRE: A simulation model for software-defined networking for communication over real-time ethernet," in *International OMNeT++ Community Summit*, 2019.

[11] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *JSAC*, 2011.

[12] T. Kobzan, S. Schriegel, S. Althoff, A. Boschmann, J. Otto, and J. Jasperneite, "Secure and time-sensitive communication for remote process control and monitoring," in *ETFA*. IEEE, 2018.

[13] R. Kumar, M. Hasan, S. Padhy, K. Evchenko, L. Piramanayagam, S. Mohan, and R. B. Bobba, "End-to-end network delay guarantees for real-time systems using SDN," in *RTSS*. IEEE, 2017.

[14] G. P. Mccormick, "Computability of Global Solutions to Factorable Nonconvex Programs: Part I – Convex Underestimating Problems," *Math. Program.*, 1976.

[15] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. Elbakoury, "Performance comparison of IEEE 802.1 TSN time aware shaper and asynchronous traffic shaper," *IEEE Access*, 2019.

[16] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software defined networks," *IEEE Transactions on Industrial Informatics*, 2017.

[17] T. Puech, M. Boussard, A. D'Amato, and G. Millerand, "A fully automated periodicity detection in time series," in *International Workshop on Advanced Analysis and Learning on Temporal Data*. Springer, 2019.

[18] S. Schriegel, T. Kobzan, and J. Jasperneite, "Investigation on a distributed SDN control plane architecture for heterogeneous TSNs," in *WFCS*. IEEE, 2018.

[19] M. Vlachos, P. Yu, and V. Castelli, "On periodicity detection and structural periodic similarity," in *International conference on data mining*. SIAM, 2005.

[20] H. Wang and M. R. Ito, "Dynamics of load-sensitive adaptive routing," in *ICC*. IEEE, 2005.

Appendix F

Towards SDN-based Dynamic Path Reconfiguration for Time-sensitive Networking

Abstract

Future networks will need to support a large number of low-latency flows. In time-sensitive networks (TSN), paths for data flows are usually established at startup time of an application and remain untouched until the flow ends. There is no way to migrate existing flows easily to alternative paths without inducing significant additional delay or wasting resources. Therefore, the resource-utilization of TSN might degrade over time leading to a sub-optimal flow assignment. In this paper we address this problem by combining Software-defined Networking (SDN) that provides better control on network flows with TSN to be able to seamlessly migrate time-sensitive flows. We propose a SDN-based dynamic path reconfiguration algorithm for accommodating TSN flows and formulate it as optimization problem. By exploiting the control plane's global view, we evaluate different dynamic path configuration strategies under deterministic communication requirements. Our simulation results indicate that reconfiguring the flow assignments from time to time can improve the latency of time-sensitive flows and can increase the number of flows embedded in the network in worst-case scenarios.

Reference

N. Sertbaş Bülbül, Doğanalp Ergenç, M. Fischer. Towards SDN-based Dynamic Path Reconfiguration for Time-sensitive Networking. IEEE/IFIP Network Operations and Management Symposium (NOMS), 2022.

Contribution

In the forementioned publication, the contribution of this thesis is designing and implementing restricted and unrestricted variations of the optimization model. The first author proposed the overall idea, implemented the other two variations in a simulation model, and conducted the evaluation. The third co-author helped to improve the quality of the paper with his valuable feedback.

Towards SDN-based Dynamic Path Reconfiguration for Time Sensitive Networking

Nurefşan Sertbaş Bülbül , Doğanalp Ergenç, Mathias Fischer
Department of Computer Science, University of Hamburg, Germany
Email:{sertbas, ergenc, mfischer}@informatik.uni-hamburg.de

***Abstract*—Future networks will need to support a large number of low-latency flows. In time-sensitive networks (TSN), paths for data flows are usually established at startup time of an application and remain untouched until the flow ends. There is no way to migrate existing flows easily to alternative paths without inducing significant additional delay or wasting resources. Therefore, the resource-utilization of TSN might degrade over time leading to a sub-optimal flow assignment. In this paper we address this problem by combining Software-defined Networking (SDN) that provides better control on network flows with TSN to be able to seamlessly migrate time-sensitive flows. We propose a SDN-based dynamic path reconfiguration algorithm for accommodating TSN flows and formulate it as optimization problem. By exploiting the control plane's global view, we evaluate different dynamic path configuration strategies under deterministic communication requirements. Our simulation results indicate that reconfiguring the flow assignments from time to time can improve the latency of time-sensitive flows and can increase the number of flows embedded in the network in worst-case scenarios.**

***Index Terms*—SDN, dynamic flow migration, reconfiguration, TSN, path computation, consistent updates**

I. Introduction

Real-time Internet of things (IoT) driven by 5G networks and autonomous vehicular networks rely on low-latency and deterministic networking. Many safety-critical applications served by such networks, e.g., robots in automation environments, require a bounded latency and a reliable delivery of data. A violation of latency constraints can, in the worst case, result in physical damage. To address real-time and deterministic communication requirements of time-sensitive and safety-critical systems, TSN standards are proposed by the IEEE 802.1 working group. TSN offers several protocols to enable the coexistence of different traffic classes with varying communication requirements in the same network.

The IEEE 802.1Qcc Stream Reservation Protocol (SRP) standard describes the management and configuration of TSN [1]. End-hosts declare their traffic requirements in the TSN before the actual communication. Then, these time-critical transmissions are scheduled to bound the undesired queuing delays, and underlying networking elements on the routing path are enforced to obey these schedules. However, new time-sensitive flows cannot be directly embedded in high traffic scenarios due to capacity limitations on certain links and the effect of link usage on latency. Hence, accommodating new flows at runtime and adapting existing flows accordingly becomes a challenging problem.

The authors of [2] show that both path splitting, i.e., sending flows over multipath and path migration lead to more flexible embeddings and better resource utilization. For that, already occupied resources need to be released by migrating flows to other paths. It requires to reconfigure the data plane while maintaining the quality of service (QoS). However, in time-sensitive networks, flow configuration is done initially, and related flow assignments remain fixed. During a reconfiguration, it is required to take down the related flow and make a new reservation, which consumes time. Therefore, reconfiguration in standard TSN networks is not efficient.

In this paper, we address the problem of dynamic path (re)configuration for TSN networks on the basis of SDN to enable the migration of network flows under strict latency constraints. The application of SDN concepts like global network view on real-time networks enables the collection and inclusion of application requirements into the configuration of network resources. Our main contributions are:

- We formulate the time-sensitive optimal routing problem (TSOR) with mixed-integer linear programming (MILP). We propose four different path configuration strategies by adding varying degrees of routing constraints to TSOR.
- Solving the MILP is not the same as a realistic evaluation in a real-time network. Apart from the number of embedded flows, the (re)configuration overhead should be considered. Therefore, we have built a realistic simulation of a TSN network in OMNeT++ and solve TSOR to obtain the optimal solution. With that we quantify the reconfiguration cost for a TSN network.
- We evaluate the presented strategies regarding the number of flows embedded into the network, reconfiguration time, and their effect on time-sensitive traffic latency. The simulation results indicate that our alternative path configuration strategies can embed more flows up to 4% without any additional delay to the time-sensitive traffic.

The remainder of this paper is structured as follows: Section II summarizes the related work on state of the art approaches for path migration problem. In Section III, we summarize our overall system and introduce the TSOR. Section IV describes our evaluation results. Section V concludes the paper and summarizes future work.

II. Related Work

Several approaches can be adapted to this domain to solve the flow migration problem, such as deploying and migrating

978-1-6654-0601-7/22/$31.00 © 2022 IEEE

virtual network functions (VNFs). In [3], the VNF mapping and scheduling problem is formalized as a mixed-integer problem (MIP) considering the VNF requirements such as delay and priority. If delay requirements are violated, they trigger delay-aware rescheduling, including the existing VNFs into the reconfiguration for a higher acceptance ratio. The authors of [4] formalize the delay-aware VNF placement and routing as an NP-hard optimization problem. Then, they solve it via an approximation, which achieves close-to-optimal performance in terms of acceptance ratio and maximum link load ratio. In [5], different approaches for a dynamic rescheduling of the placement of VNFs are proposed. The re-optimization approach strictly preserves latency constraints by being triggered at every time instance but requires many VNF migrations. Alternatively, they also propose time-triggered re-optimization with either fixed or dynamically updated (depending on the network resources and controlled by the operator) trigger times. In [6], the authors propose a MILP optimization model to decide between either migration of VNFs or their re-instantiation. In addition to VNF related constraints such as availability of VNFs' services, maximum delay, and maximum resource consumption, e.g., memory and CPU, they also consider the update time for the data plane as a convergence constraint derived from SDN.

Our work extends these concepts and maps them to the real-time flow migration problem with more strict real-time constraints. If VNFs are in the data path, the migration of stateful VNFs requires additional mechanisms and protocols to keep the states throughout the migration process. However, in this study, we focus on the flows only.

There is limited literature focusing on the incrementally adding flows in TSN. However, most related papers assume that routing paths are known apriori and left the path (re)configuration out of scope. In [7] authors propose an SDN-based resource allocation mechanism for accommodating new flows at runtime. They also propose an indirect path migration algorithm in case direct path migration is not feasible. However, they focus on the feasibility of migration, and migration overhead (e.g., end-to-end latency and number of reconfigured paths) is not analyzed. Also, their flow migration definition only involves routing path changes; does not take schedule changes into account. Since TSN is designed to isolate flows either spatially through different routes or temporally through different schedules, separating routes from schedules may limit the QoS. In contrast to the related work so far, few publications address scheduling together with the path computation problem [8], [9]. In [10], an SDN-based self-configuration framework for TSN networks has been presented. The central SDN controller, initially puts streams to the default paths and then moves streams to optimal paths based on the extracted stream characteristics. To find optimal paths that maintain hard real-time guarantees, not only the path length but also the latency deriving from the schedule configurations are taken into account.

Our work partially intersects with these studies by combining routing paths with schedules. However, as in some of the mentioned studies, we do not compute time-division multiple access-based schedules in which multiple frames are transmitted one after the other, each using its time slot. Instead, we embed the gate opening frequency into our path computation formalization as we explain in the following section. Moreover, unlike the existing optimization models that solve path assignment problems with rather simple metrics such as path lengths and link weight, our model includes the gate configuration as a TSN-specific aspect. We defined the optimal routing problem together with the service-based stream configuration regarding the main characteristics of TSN. At that point, we differ from the related work.

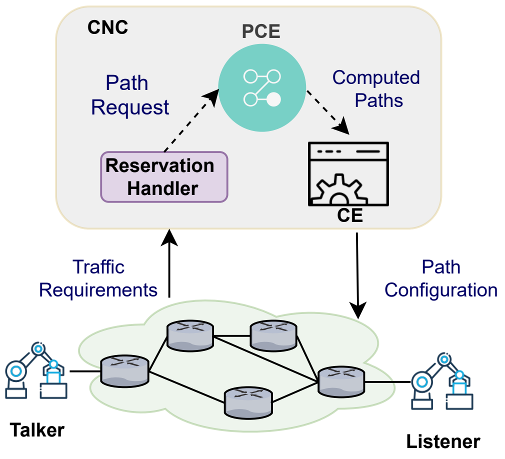


Fig. 1. Overall system block diagram

III. System Design

In this section, we introduce our SDN-based dynamic reconfiguration solution for TSNs. We first describe the overall framework, and afterward, we explain four different path configuration strategies in detail.

A. Overall System

We propose a (re)configuration framework for the time-sensitive networks by benefitting from the SDN. The global view of the centralized SDN controller enables the deployment of centralized routing algorithms and eases the configuration. Thus, routing paths could be reconfigured dynamically on the runtime considering the requirements of the time-sensitive environments.

We illustrate our SDN-based framework in Fig. 1. To communicate in such a network, the end-host, a talker in TSN, needs to inform the network to allocate required resources before the actual communication. The talker initiates that process by sending a talker-advertise message to the edge switch. Then, the edge switch forwards the traffic requirements of the flow, which are obtained from the talker-advertise message to the SDN-supported *centralized network controller (CNC)*. The global network view of the CNC enables efficient use of network resources and fast responses to varying network conditions. Here, the *reservation handler* records the

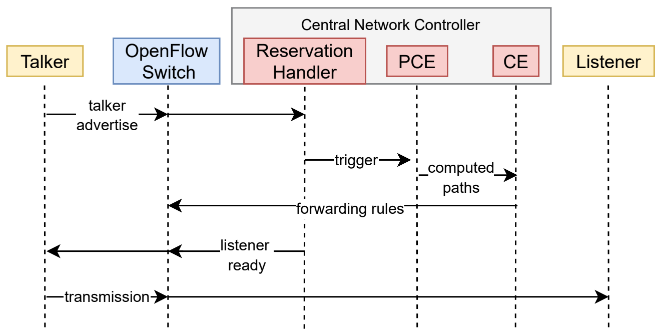


Fig. 2. Time sequence of the overall system

received request and triggers the *path computation element (PCE)*. Then, PCE computes a new path considering the traffic requirements, current resource utilization, and the topology. The computed path is sent to the *configuration engine (CE)*, and related forwarding rules are distributed via the data plane, guaranteeing the consistency of the data plane.

In some cases the computed path may not be free and requires the migration of existing flows. Here, CE migrates flows sequentially, ensuring consistency, and then the new flow is accommodated. After all forwarding rules are successfully updated at the respective switches, the reservation handler sends a listener-ready message to the talker. Then, since required resources for the transmission have already been provided, the talker starts to send data via the allocated path. This procedure has been illustrated in the time chart in Fig. 2.

B. Path Computation Engine

In the following, we present the time-sensitive optimal routing (TSOR) problem with varying degrees of routing constraints. Removing such constraints from the model improves the solution quality regarding accommodating flows while adding computational complexity. Figure 3 illustrates the flowchart for the path computation and configuration processes. If all constraints can be satisfied, PCE returns with a solution that may require changes in the previous flow assignments. Otherwise, it rejects the flow without embedding.

1) Problem Formulation: We formulate TSOR as a MILP model to migrate high-priority flows to suitable paths. Using the model, we find (i) end-to-end paths for given demands under different QoS requirements within limited network resources and (ii) gate configurations for each switch that minimizes the overall end-to-end communication latency.

The gate configuration is the primary mechanism of the core TSN protocol, 802.1Qbv Time-aware Shaper (TAS) that ensures end-to-end deterministic communication for the streams of different QoS classes via strict time-division scheduling [11], [12]. In TAS, on each (egress) port of a switch, there are eight priority queues that store frames of streams with different priorities, including best-effort, before they are forwarded to the destination. Each queue is controlled by a gate to forward a frame. When a gate is open, the next frame in the respective queue is sent at a given time. Eight gates corresponding to the eight priority classes are configured by a gate driver via a gate control list (GCL) that decides which gate(s) should be open at which time. This mechanism overall constitutes a frame-forwarding schedule with respect to the priority classes to satisfy strict timing requirements.

$$\min \sum_{d\in D}\sum_{p\in P_d}\sum_{e\in E} x_{dp}\alpha_{ep}\left[l_e^o + l_e^q(1-g_{es})\right] \tag{1}$$

$$\sum_{p\in P_d} x_{dp} = 1 \qquad \forall d \in D \tag{2}$$

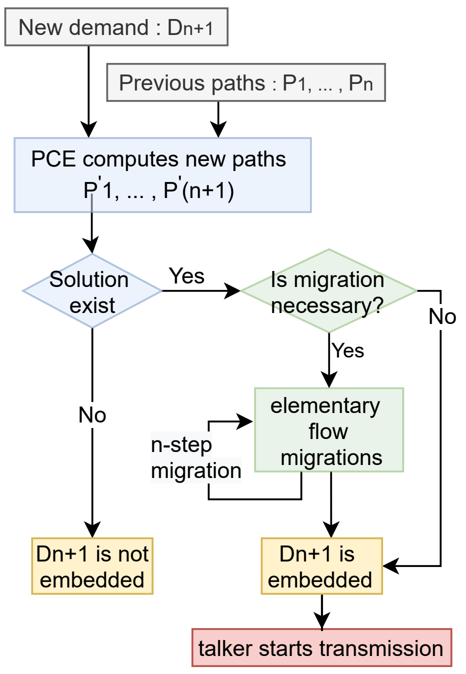
$$\sum_{d\in D}\sum_{p\in P_d} x_{dp}\alpha_{ep}h_d \le c_e \qquad \forall e \in E \tag{3}$$

$$\sum_{s\in S} g_{es} = 1 \qquad \forall e \in E \tag{4}$$

$$\sum_{p\in P_d}\sum_{e\in E} x_{dp}\alpha_{ep}\left[l_e^o + l_e^q(1-g_{es})\right] \le l_d \qquad \forall d \in D \tag{5}$$

$$g_{es} - \sum_{d\in D}\sum_{p\in P_d} x_{dp}\alpha_{ep}\frac{h_d}{c_e} \ge 0 \qquad \forall e \in E, \forall s \in S \tag{6}$$

The formulation of TSOR is given in Equations 1-6. Table I also summarizes all the variables and parameters used in the model. To formalize our problem, we utilize two optimization variables: x_{dp} and g_{es}. x_{dp} is a binary variable to decide if demand $d \in D$ is assigned to directed path $p \in P_d$. Here, each d is defined between a talker and a listener, where D is the set enumerating all demands. Accordingly, P_d represents



New demand : Dn+1
Previous paths : P1, ... , Pn
PCE computes new paths P'1, ... , P'(n+1)
Solution exist
Yes
Is migration necessary?
No
Yes
No
elementary flow migrations
n-step migration
Dn+1 is not embedded
Dn+1 is embedded
talker starts transmission

Fig. 3. Flowchart for the path configuration

TABLE I

TERMS AND DEFINITIONS IN THE OPTIMIZATION PROBLEM. *Base* TYPE CONTAINS THE FUNDAMENTAL ELEMENTS OF THE MODEL. *Constant*S ARE NETWORK- AND SERVICE-RELATED PARAMETERS GIVEN AS INPUT. *Variable*S REPRESENT THE PARAMETERS TO BE OPTIMIZED.

Type	Symbols	Set	Interval	Definition
Base	d	D		A demand between a pair of nodes
	p	P_d		A (candidate) path to be assigned to demand d
	e	E		A link (edge) between nodes
	s	S	[0, 7]	A quality of service class
Constant	c_e	$\Re^*$	[0,∞]	Maximum link capacity of e
	h_d	$\Re^*$	[0,∞]	Traffic volume of d
	α_{pe}	$\Re^*$	[0,1]	Binary variable to indicate if link e belongs to path p
	l_d	$\Re^*$	[0,∞]	Latency requirement of d
	l_e^q	$\Re^*$	[0,∞]	Queueing delay factor on link e
	l_e^o	$\Re^*$	[0,∞]	Default latency on link e due to port and link characteristics
	a_{dp}	Z^*	[0,1]	Binary variable to indicate if demand d allocated to path p in the previous configuration
Variable	x_{dp}	$\Re^*$	[0,∞]	Binary variable to decide if demand d allocated to path p
	g_{es}	Z^*	[0,1]	Opening frequency of the gate for service class s on link e

the set of paths computed between those two particular end-points. g_{es}, is a continuous variable defined within $[0, 1]$ and represents the frequency of an open gate on the egress port of link $e \in E$ for the service class s among eight possible classes. Thus, g_{es} specifies the priority given to service class s on a directed link e. While $g_{es} = 1$ infers that the gate for s should be open all the time and the capacity of the entire link e is used for that type of demands, $g_{es} \approx 0$ means that any demand of service type s is not active at all on the respective port and thus, the gate is closed. Otherwise, the respective gate for the service class s on link e is open as proportional to $0 < g_{es} < 1$. From this perspective, g_{es} is affected by the total required resources for the demands of service type s as the available capacity, e.g., bandwidth, of e is distributed among those demands according to their service type. Note that each demand is associated with a particular service class randomly and given as an input.

Our objective function (1) minimizes the overall latency of the selected paths. The variables in the objective function are explained in detail in the context of the latency constraint (5). Constraint (2) ensures that each demand $d \in D$ is assigned to exactly one path $p \in P_d$. Here, we assume that all flows are non-bifurcated. Constraint (3) guarantees that each link e has sufficient capacity c_e to handle the total load h_d of all demands $d \in D$ assigned to any path p including e, s.t. $\alpha_{ep} = 1$. Constraint (4) represents the configuration of the gate control list of e for each class of service s. As the gates, i.e., enabling queues of an egress port, share limited link resources, only a set of them can be practically open at the same time but proportional to the value of g_{es}. Constraint (5) ensures that the end-to-end latency on path p is always smaller than the maximum allowed latency for demand d, which is l_d. Besides, the gate configuration g_{es} on the respective egress port of each link e that belongs to path p, s.t. $\alpha_{ep} = 1$, impacts the end-to-end latency. Constraint (6) forces g_{es} to be proportional to the total traffic load of service type s forwarded through link e. Note that while higher values of g_{es} positively impact the latency at link e, as it enables the traffic of service type s more often, a smaller g_{es} causes an increased latency due to queueing delay in the respective gate. Accordingly, we add the delay factor l_e^q to the proportion $1 - g_{es}$ to represent the queueing delay. Apart from that, a base delay l_e^o representing the port and link characteristics, e.g., packet processing and propagation delay, is considered for each link. While those design parameters, l_e^q and l_e^o, can be set according to the system and network properties, we use $l_e^q = 0.5$ and $l_e^o = 1.0$ in our simulations.

$$x_{dp} \geq a_{dp} \qquad \forall d \in D, \forall p \in P_d \tag{7}$$

Constraint (7) is the pre-assignment constraint that fixes the demands that are already assigned to a certain path p, i.e., $a_{dp} = 1$ from an existing configuration. a_{dp} is given as input to the problem. For instance, when a new flow has to be scheduled, the former configuration with existing flows can be held intact to find a new path with a suitable gate configuration only for the new flow. Note that although keeping the previous demands fixed before allocating a new demand reduces the flexibility of routing, it is important to have a stable configuration scheme especially for the critical and high-priority demands. That is, reconfiguring the network has also a certain cost, e.g., delay for migrating flow, sending control packets to the switches, and can hinder the deterministic communication requirements. Therefore, enabling reconfiguration by flow migrations requires to involve such costs in the end-to-end latency. The use of the preassignment constraint is discussed further in the following section.

Considering the complexity, TSOR has $\mathcal{O}(|D||P| + |E|)$ optimization variables where the number of paths are directly related to the number of links. Note that even though g_{es} depends on the number of service classes, it is, at least in TSN context, defined as eight (including best-effort) and thus we assume that as a constant. In terms of the number of constraints, TSOR is bounded by $\mathcal{O}(|D||P| + |E|)$ constraints with the same assumption on the number of services. Another important complexity issue is the non-linear constraints and the objective function. It is easily possible to linearize the multiplication of a binary variable x_{dp} and non-binary variable g_{es} using, for instance, McCormick envelopes [13] introducing some additional complexity. Therefore, we take TSOR as a linear problem that makes it more convenient to be solved by the state-of-the-art linear optimization tools.

2) Path Configuration Strategies: Incremental flow scheduling in TSN will change the link and switch utilization over time, affecting the end-to-end latency of chosen paths. Thus, we present different path configuration strategies with varying degrees of routing constraints.

a) Reconfiguration at every path request: To maximize the number of flows embedded via TSOR, replanning all path configurations from scratch is a strategy. For that, we remove the pre-assignment constraint from TSOR and present *unrestricted* version as **TSOR-U**. With that, we allow all flows to be reconfigured, e.g., migrated to different paths changing the gate configuration as well, to find the optimal allocation, including newly arriving flows, with a certain cost of reconfiguration. Here, the cost includes the delay of frames due to the control packets exchanged between the controller and switches to configure the data plane for each migrated flow. Therefore, even though it can flexibly configure the network resources, it introduces additional configuration overhead, which may hinder the deterministic communication requirements. To address that issue, we also present the following strategies:

b) Restricted path reconfiguration: The strict time constraints of such time-sensitive environments lead to the accommodation of flows on certain paths and leave these paths untouched as long as the path meets the delay requirements. Therefore, we have a pre-assignment constraint in our optimization problem that keeps the previous assignments fixed. We name this *restricted* version of our optimization problem, **TSOR-R**. Although such a constraint reduces the flexibility of routing, it is important to have a stable configuration scheme, especially for critical and high-priority demands.

c) Reconfiguration at every k-th path request: To shorten the time for finding paths, we can still use *TSOR-R* for embedding new flows. However, this will lead to inefficient use of resources, especially for a larger number of flows. Therefore, another solution is to reconfigure the assignments from time to time to ensure better resource utilization. For that, we propose **TSOR-P** that reconfigures the network after having received k flow requests. Therefore, it can adapt the reconfiguration period dynamically in dependence on the arrival rate of flows. This strategy can also be improved by monitoring the system and extracting a pattern for the latency violations to compute optimal reconfiguration times.

d) Threshold-triggered reconfiguration: The most straightforward strategy **TSOR-T** a network operator can apply is to use *TSOR-R* to embed a new flow and to compute *TSOR-U* to measure how close the resulting solution is to *TSOR-U*. Then, we only reconfigure if the computed objective exceeds a pre-defined threshold.

We use Fig. 4 to describe two scenarios for explaining the difference between our path configuration strategies. For simplicity, we have three flows, f_1, f_2, and f_3 with the same size, and each link has one flow capacity. Suppose initially only f_1 is routed in the network as in Fig. 4-a. Two different situations can arise for new arriving flows, either (b) or (c):

In the first case, f_2 flow arrives and assume that the computed optimal path for f_2 is S2 to S4, which is not used by other flows. Therefore, f_2 can be directly placed on this path as seen in Fig. 4-b. Here, all path configuration strategies produce this selection as an optimal solution without a flow migration.

In the second case, let us assume that f_3 flow arrives and the computed optimal path for f_3 is S3 to S4, which is already occupied by flow f_1. Here, the migration of f_1 is required to another path (e.g., S1-S2-S4), and this can happen directly as in Fig. 4-c and then the link between S3 to S4 becomes free. With *TSOR-U*, since it reconfigures the network from scratch, it finds the solution by migrating f_1 to its new path first and then accommodating f_3. However, since *TSOR-R* does not allow for a reconfiguration, it does not accommodate f_3 in the network and reject. It depends on the current network state whether *TSOR-T* and *TSOR-P* can find a solution for f_3 or not.

C. Configuration Engine

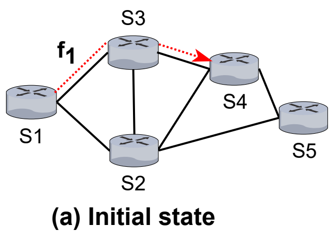
The CE is responsible for applying the flow rules for the paths generated by the PCE consistently. Since all switches may not be updated simultaneously, changes in the network configuration may cause incorrect forwarding behavior and performance disruptions. In literature, this is known as network update problem [14]. To ensure correct forwarding behavior when changing the flow assignments, we use a two-phase tagging mechanism proposed in [14]. With that, both the initial and final rules are installed on all switches, and the packets are tagged with the version number of the respective forwarding rule. All switches on the path are updated with the new flow rules, and each updated switch sends an ACK to the controller. When the controller has received an ACK from all related switches, packets entering the network are tagged with the new version number to match the new flow rule. As a drawback, this mechanism doubles the number of flow entries on switches. However, this could be solved by regularly deleting old rules.

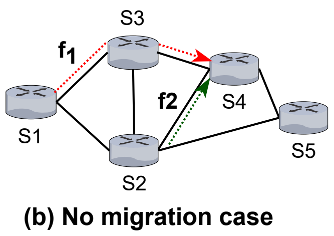
IV. Evaluation

In this section, we evaluate the reconfiguration overhead for time-sensitive traffic by applying the strategies introduced in Section III-B. First, we briefly explain the evaluation setup and metrics. Then, we evaluate the performance of the proposed strategies at varying traffic loads. Finally, we summarize our evaluation results.

A. Experimental Setup

To measure the reconfiguration overhead and its effects on the time-critical communication, solving MILP-based path configuration as standalone is insufficient. Thus, we implemented the presented path configuration strategies in CPLEX 12.7.0 and simulated the TSN network in OMNeT++ v5.5.1. For that, we used the INET framework and extended the





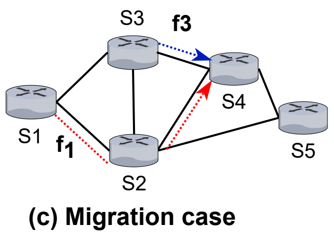


Fig. 4. Example scenario

SDN4CoRE framework [15] that enables the configuration of SDN and TSN capable switches. We developed three applications for the control plane: a reservation handler module, a path computation element, and a configuration engine, whose details are presented in Section III.

For our experiments, we used the *Integra* topology, as represented in Fig. 5, whose average node degree is 2.67, from the Topology Zoo dataset [16]. It contains 27 switches and 36 edges. We consider all nodes in the topology to be OpenFlow-enabled TSN switches. Since the relative results are not changed, we set the link capacity to 30Mbps for the sake of simplicity.

Since there is no publicly available data set for TSN traffic, we obtained the traffic generation parameters from TSN papers and tried to model TSN traffic as realistic as possible.

As time-triggered (TT) traffic, we used cyclic traffic and select the transmission period uniformly between 2-20 ms and a data size between 50 and 1000 bytes as defined in [17]. Thus, the data rate can take values in between [0.02-4.0] Mbps. We also generated traffic using pareto, uniform, and normal distributions in the given data rate range. With that, we tried to evaluate in more diverse traffic scenarios. In pareto distribution, a large portion of the generated traffic has low data rates. In contrast, uniform and normal distributions represent medium data-rated applications in this setup. A typical application for the cyclic traffic is the input/output updates exchanged between actuators, sensors, and PLCs in an industrial facility.

Since different service classes coexist in the same TSN network, we also generated best-effort (BE) traffic that has no timing guarantee. For that, we set the packet size to 1500 bytes and use exponentially distributed packet inter arrival times [12]. We set the same packet generation rate at each BE traffic source and start BE traffic at the beginning of the simulation. A typical example for the BE traffic can be retrieving application data (e.g., telemetry). Table II summarizes the simulation parameters.

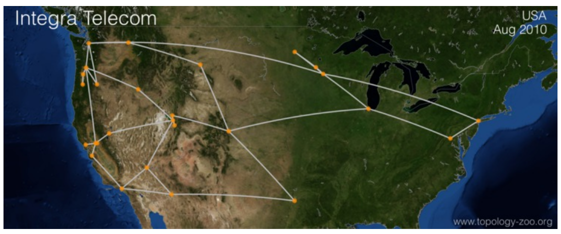


Fig. 5. The Integra topology used in the evaluation [16]

TABLE II
SIMULATION PARAMETERS

Category	Parameter	Value
Time-triggered traffic	Transmission period	Uniform(2,20) ms
	Frame size	50-1000 bytes
	Data-rate distribution	pareto, uniform, normal
Best-effort traffic	Transmission period	Exponentially distributed
	Frame size:	1500 bytes
Topology	Num. of switches	27
	Num. of edges	36
	Link capacity	30Mbps
Simulation	Duration	50 sec

B. Evaluation Metrics

We used the following metrics to evaluate our path configuration strategies:

- **Acceptance Ratio:** The ratio of TT flows that can be successfully served by the network.
- **Missing Deadline Ratio:** The ratio of delayed frames whose delay exceeds the delay requirement to the received frames.
- **Reconfiguration Ratio:** The ratio of the number of paths changed during reconstruction to the number of flows.
- **Configuration Time:** The time before the actual communication starts. Thus, it includes the potential migration latency and the tag-based data plane configuration time.

C. Results

This section compares the path configuration strategies in terms of their acceptance ratio and reconfiguration overhead.

Acceptance Ratio We measured only the acceptance ratio of *TSOR-R* and *TSOR-U* to draw the limits of our optimization problem. Since different data rates affect how flexible TSOR can accommodate flows, we measured the average acceptance ratios independence on different data rate distributions as shown in Fig. 6.

The experiments were repeated 100 times for each scenario, and the results are given with a 95% confidence interval. Here the acceptance rate does not differ a lot thus the confidence intervals are very small. Since *TSOR-U* can fully utilize all data plane resources, it has higher acceptance ratio on the

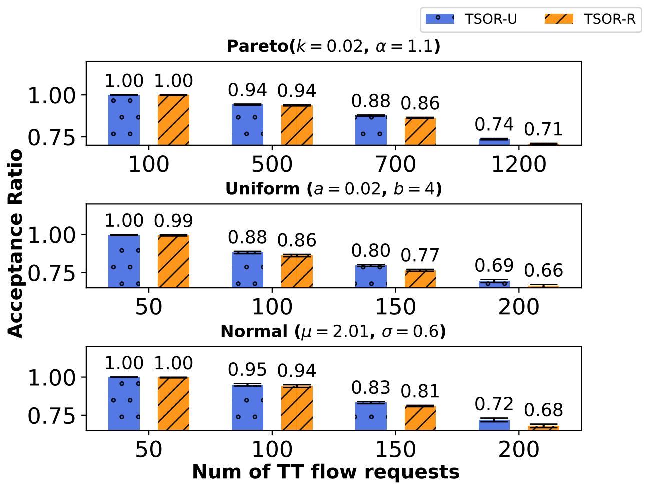


Fig. 6. Acceptance ratio with varying data rate distributions

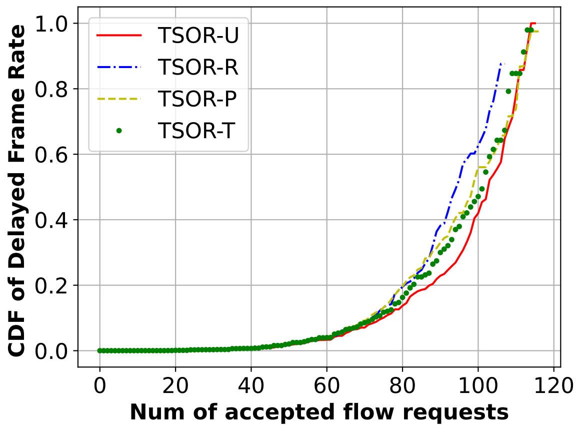


Fig. 7. Missing deadline ratio respect to number of accepted flows

average in all distributions. However, *TSOR-R* leaves less room to accommodate new flows, as the previously installed flows are not touched. So this limits the acceptance ratio of *TSOR-R*. Also, the difference between *TSOR-R* and *TSOR-U* becomes more visible when the number of TT flow requests increases due to limited flexibility of *TSOR-R*. The difference becomes more significant around 4% in some cases (e.g., 200 TT flows case in normal distribution). Also, between the distributions we see that the lower data rates as in pareto allow accommodating more flows than other medium-sized distributions, which are uniform and normal in this setup.

Reconfiguration Overhead To measure the reconfiguration overhead and its effects on the delivery of TT traffic, we simulated a TSN network in OMNeT++. Since the relative results do not change significantly for different distributions, we generated 200 normally distributed TT and 50 BE flows. Then, we measured the reconfiguration overhead with presented evaluation metrics.

Table III shows the simulation results for TSOR. Note that, even though we simulated also BE traffic in the network, we do not count it while we compute accept ratio. For that reason, accept ratios seems lower here. We measured the missing deadline ratio, which is mainly related to the quality of chosen paths. Since *TSOR-R* cannot flexibly change the assigned paths based on the current network status, some links may be overloaded while there are spare link resources. Thus, it has

TABLE III
OMNeT RESULTS WITH BEST-EFFORT TRAFFIC

	TSOR-U	TSOR-R	TSOR-P	TSOR-T
Acceptance Ratio [%]	52.81	49.2	51.18	51.93
Missing Deadline Ratio [%]	5.41	5.83	4.94	4.47

the highest missing deadline ratio. Here the first expectation is that *TSOR-U* has the lowest missing deadline ratio since it can flexibly use the resources. However, there is a hidden effect: the network utilization directly influences the delay, but the impact gets more significant the more utilized a network is. In other words, since *TSOR-U* can embed more flows, the network load and therefore the average TT latency increases, which also increases the missing deadlines.

To highlight this, we also plotted the cumulative density function of the missing deadline ratio in dependence on the number of accepted flows in Fig. 7. Here, for the same number of accepted flows, we can see that *TSOR-U* has the lowest delayed frame rate, which supports our claim. In *TSOR-P*, we set k to 20, so that reconfiguration is triggered for every 20th received request. Therefore, it can adapt resources depending on the received traffic rate. In *TSOR-T*, reconfiguration is only triggered if the solution quality in terms of latency exceeds a certain threshold, e.g., 1%. Thus, both *TSOR-P* and *TSOR-T* perform better than *TSOR-R* and are close to *TSOR-U*. However, after a certain point, which is around 110 flows in this experiment, the number of delays increases significantly in *TSOR-U*. Thus, even though it has the lowest missing deadline ratio until there, it will not performs better than the *TSOR-P* and *TSOR-T* after that point.

Fig. 8 shows the reconfiguration overhead of TSOR versions for the same number of flows. In Fig. 8 top, *TSOR-R* has the smallest reconfiguration ratio, i.e., zero, since it does not allow reconfiguration. However, *TSOR-U* has the highest due to frequent reconfigurations. In *TSOR-P*, the number of reconfigurations is directly related to the number of flows and independent of the current network status. Therefore, it increases with a higher number of flows. Even though *TSOR-T* is triggered per received request, the thresholding mechanism avoids unnecessary reconfigurations. However, the performance of *TSOR-P* and *TSOR-T* is highly related to the chosen parameters. Decreasing the threshold in *TSOR-T* and

TABLE IV
PATH (RE)CONFIGURATION STRATEGIES

Strategy	Reconfiguration Trigger	Flexibility	Time-sensitive traffic latency	Configuration Overhead
TSOR-U	After every critical flow	High	Medium	High
TSOR-R	No reconfiguration	Low	High	Low
TSOR-P	After every k-th critical flow	Medium	Low	Medium
TSOR-T	After every critical flow that requires sufficient* changes	Medium	Low	Medium

k-value in *TSOR-P* will approximate solutions to the *TSOR-U*.

We defined the configuration time as the potential migration latency and the data plane configuration time. Thus, it is directly affected by the number of reconfigurations and reconfigured switches on the path. Therefore, we see the similar results in Fig. 8 bottom, which shows the configuration time of the TSOR. Frequent reconfigurations in *TSOR-U* increase configuration time, while limited reconfigurations in *TSOR-R* result in lower configuration time. As in the Fig. 8 top, both *TSOR-P* and *TSOR-T* performs in between *TSOR-R* and *TSOR-U*.

In our simulations we excluded the time to solve the MILP optimization models, i.e., they were solved in zero time, to fairly compare the different embedding methods. However, the actual time required to solve the MILP is still in a reasonable range, e.g., around 7.3s for 100 flows. This is a significant overhead for TSN, but only in TSOR-U the MIP needs to be solved for every new flow. TSOR-R builds upon the previous solution. Thus, sorting in a new flow for TSOR-R requires only around 4 ms in our settings. The other strategies (TSOR-T/P) require to solve the optimization model for all flows from time to time. They will then migrate embedded flows from their potentially sub-optimal paths to their optimal ones. Even tough their performance is highly related to the configuration parameters such as k in *TSOR-T* and the triggering threshold in *TSOR-P*, this migration can be done seamlessly without packet loss and increased latencies. We summarize our path reconfiguration strategies in Table IV.

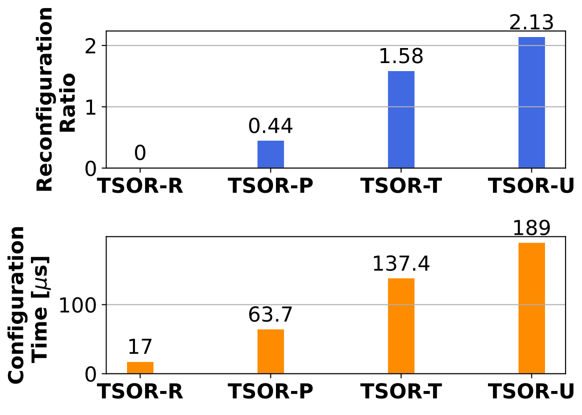


Fig. 8. Reconfiguration overhead for TSOR

V. CONCLUSION

This paper presents and evaluates dynamic path configuration strategies for SDN-enabled time-sensitive networks. We defined a restricted optimal flow placement model that adapts path assignments based on the current resource utilization. Then, we present three heuristics to maximize the number of accepted flows while meeting the communication requirements of TT applications. Our restricted optimization model yields the results in terms of configuration time and serves as a benchmark for other heuristic solutions.

We believe that a proper reconfiguration strategy can be selected depending on the requirements of the environment. For example, in a highly dynamic small or medium scale environment where flows are added and removed over time, *reconfiguring at every path request* would be more appropriate for utilizing all resources more efficiently. Our simulation results indicate that it increases the number of flows that can be embedded by up to 4%. However, it may not be desired for large-scale networks to migrate flows that frequently. Alternatively, *reconfiguring at every k-th path request* and *reconfiguring only when the solution deviates more than a threshold* from the optimal solution achieve a larger number of accepted flows at moderate configuration overhead. For that, the parameter selection plays an important role. The selection of lower k values and reconfiguration thresholds increases the reconfiguration frequency. Therefore, they appear as promising reconfiguration solutions for time-sensitive scenarios.

We envision that due to failures or dynamic traffic patterns, flow migration would be triggered more often. However, in such cases the performance becomes a significant design criteria, especially for time-sensitive networks. Therefore, we showed the feasibility of such migrations in real-time with this paper. As future work, we would like to include different aspects, e.g., minimizing the required flow migrations and splitting flows into multiple paths. Such aspects enable more balanced flow placement with better resource utilization.

REFERENCES

[1] "IEEE standard for local and metropolitan area networks–bridges and bridged networks - amendment 31: Stream reservation protocol enhancements and performance improvements," 2018.
[2] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 17–29, 2008.
[3] J. Li, W. Shi, P. Yang, and X. Shen, "On dynamic mapping and scheduling of service function chains in sdn/nfv-enabled networks," in *IEEE GLOBECOM*, 2019, pp. 1–6.

[4] S. Yang, F. Li, S. Trajanovski, X. Chen, Y. Wang, and X. Fu, "Delay-aware virtual network function placement and routing in edge clouds," *IEEE Transactions on Mobile Computing*, vol. 20, pp. 445–459, 2021.

[5] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vnf placement at the network edge," in *IEEE INFOCOM*, 2018.

[6] H. Hawilo, M. Jammal, and A. Shami, "Orchestrating network function virtualization platform: Migration or re-instantiation?" in *IEEE CloudNet*, 2017.

[7] P. Danielis, G. Dán, J. Gross, and A. Berger, "Dynamic flow migration for delay constrained traffic in software-defined networks," in *IEEE GLOBECOM*. IEEE, 2017, pp. 1–7.

[8] A. Alnajim, S. Salehi, and C.-C. Shen, "Incremental path-selection and scheduling for time-sensitive networks," in *IEEE GLOBECOM*, 2019.

[9] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software-defined networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2066–2075, 2017.

[10] N. S. Bülbül, D. Ergenç, and M. Fischer, "SDN-based self-configuration for Time-Sensitive IoT Networks," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, 2021, pp. 73–80.

[11] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," pp. 1–57, 2016.

[12] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. Elbakoury, "Performance comparison of ieee 802.1 TSN time aware shaper (TAS) and asynchronous traffic shaper (ATS)," *IEEE Access*, vol. 7, pp. 44 165–44 181, 2019.

[13] G. P. Mccormick, "Computability of Global Solutions to Factorable Nonconvex Programs: Part I – Convex Underestimating Problems," *Math. Program.*, 1976.

[14] D. Li, S. Wang, K. Zhu, and S. Xia, "A survey of network update in SDN," *Frontiers of Computer Science*, vol. 11, no. 1, pp. 4–12, 2017.

[15] T. Häckel, P. Meyer, F. Korf, and T. Schmidt, "SDN4CoRE: A simulation model for software-defined networking for communication over real-time ethernet," in *International OMNeT++ Community Summit*, 2019.

[16] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[17] A. Ademaj, D. Puffer, D. Bruckner, G. Ditzel, L. Leurs, M.-P. Stanica, P. Didier, R. Hummen, R. Blair, and T. Enzinger, "Industrial automation traffic types and their mapping to QoS/TSN mechanisms," 2019.

Appendix G

On the Reliability of IEEE 802.1CB FRER

Abstract

The introduction of IEEE Time-sensitive Networking (TSN) enables the design of real-time and mission-critical networks based on Ethernet technologies. Apart from providing necessary tools for near-deterministic scheduling, TSN comes with further functionalities for configurability, security, and reliability. IEEE 802.1CB Frame Replication and Elimination (FRER) is the only protocol in the TSN toolbox for adding fault-tolerance via sending the same packets via redundant paths. Although its core functions are defined by the standard, its effective use mainly depends on the actual deployment scenario and the path selection strategy. In this paper, we show that FRER can induce unintentional elimination of packets packets when the paths chosen for a particular packet flow are non-disjoint. We propose the new metric *reassurance* that can be used in FRER path selection. Besides, we propose an additional enhancement to FRER that can prevent unintended packet eliminations independent from the deployment scenario. Our simulation results indicate that the reassurance-based path selection performs better than random or maximum-disjoint path selection in random failure scenarios.

Reference

```
Doğanalp Ergenç, M. Fischer.  On the Reliability of IEEE 802.1CB
FRER. IEEE International Conference on Computer Communications (IN-
FOCOM), 2021.
```

Contribution

In the forementioned publication, the whole contribution belongs to this thesis. The co-author helped to improve the quality of the paper with his valuable feedback.

On the Reliability of IEEE 802.1CB FRER

Doğanalp Ergenç
Universität Hamburg, DE
ergenc@informatik.uni-hamburg.de

Mathias Fischer
Universität Hamburg, DE
mfischer@informatik.uni-hamburg.de

Abstract—**The introduction of IEEE Time-sensitive Networking (TSN) enables the design of real-time and mission-critical networks based on Ethernet technologies. Apart from providing necessary tools for near-deterministic scheduling, TSN comes with further functionalities for configurability, security, and reliability. IEEE 802.1CB Frame Replication and Elimination (FRER) is the only protocol in the TSN toolbox for adding fault-tolerance via sending the same packets via redundant paths. Although its core functions are defined by the standard, its effective use mainly depends on the actual deployment scenario and the path selection strategy. In this paper, we show that FRER can induce unintentional elimination of packets packets when the paths chosen for a particular packet flow are non-disjoint. We propose the new metric *reassurance* that can be used in FRER path selection. Besides, we propose an additional enhancement to FRER that can prevent unintended packet eliminations independent from the deployment scenario. Our simulation results indicate that the reassurance-based path selection performs better than random or maximum-disjoint path selection in random failure scenarios.**

Index Terms—**time-sensitive networks, fault-tolerance, FRER**

I. Introduction

From autonomous cars to smart factories, modern safety critical-systems consist of a multitude of interconnected components such as sensors, processors, and embedded controllers. Those components are distributed over the system and require a reliable, efficient, and mostly time-sensitive communication service to operate in concert. In different domains, specifically tailored networking equipment and dedicated protocols are being used to satisfy various communication requirements. For instance, Controller Area Network (CAN), Local Interconnected Network (LIN), and Media Oriented System Transport (MOST) are designed for an application in the automotive domain with its latency and bandwidth requirements [1]. Avionics Full Duplex Ethernet (AFDX) is used and standardized for avionics to satisfy similar needs [2]. As a result of such diversity across different domains, there has been a lack of generic networking equipment and protocols for safety-critical networks. To overcome this, Time-sensitive Networking (TSN) standards are recently proposed by the IEEE 802.1 Working Group to address real-time and deterministic communication requirements of time-sensitive and safety-critical systems on top of the IEEE 802.3 Ethernet standard [3]. TSN offers a spectrum of protocols to manage different traffic classes, ensure deterministic communication within a bounded delay, define filtering and networking policies, and improved reliability by using redundant paths. Moreover, TSN allows to use commodity-off-the-shelf hardware, i.e., Ethernet switches supporting TSN protocols.

802.1CB Frame Replication and Elimination for Reliability (FRER) is the primary solution for TSN to tolerate link and node failures [4]. For that, it offers a static redundancy mechanism by replicating each packet, i.e., Ethernet frame, via multiple, preferably node-disjoint redundant paths. FRER also comes with an elimination mechanism that runs on relay systems, i.e., Ethernet switches, and end-hosts to drop replica packets. This mechanism deletes redundantly received packets and protects the network against loops and babbling idiots, e.g., stuck senders sending the same packets unintentionally.

When disjoint paths are not possible, finding paths that maximize fault-tolerance should be the main concern. As the replica elimination of FRER drops redundant packets, i.e., packets with the same sequence number, a relay used in two paths will drop one replicated flow of packets. Therefore, poorly selected paths can inhibit both communication reliability and efficiency. Although FRER provides redundancy via multiple paths, selecting those paths are mostly a matter of configuration handled by an external network discovery mechanism such as 802.1Qca Path Control and Reservation (PCR) [5], which is an extension of 802.1aq Shortest Path Bridging (SPB) [6] and leverages the Intermediate System to Intermediate System (IS-IS) routing protocol. In this paper, we discuss the main issues of FRER and propose different approaches to use it effectively to ensure reliability. Accordingly, our contributions are:

- We have analyzed FRER and have identified a shortcoming in its packet elimination mechanism that results in lost packets at the destination in cases where non-disjoint redundant paths are used.
- We propose a new graph metric, *reassurance*, to find redundant paths between given endpoints that maximize fault-tolerance in case of random node and link failures. The metric enables selecting suitable paths, independent from the FRER stack design.
- We enhance the FRER sequence recovery function, to select better paths and to avoid unintended packet eliminations.

The rest of this paper is structured as follows. Section II introduces the basics of FRER and describes the main problem with its elimination mechanism. Section III presents *reassurance* as a new graph metric to be used as an effective path selection strategy. In Section IV, we explain our FRER enhancements to overcome the described problem. In Section V, we compare the reassurance-based path selection with other path selection strategies and show the efficiency of our

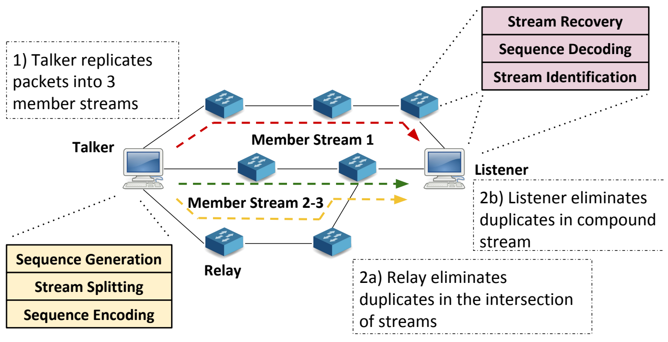

1) Talker replicates packets into 3 member streams
Stream Recovery
Sequence Decoding
Stream Identification
Talker
Member Stream 1
Listener
Member Stream 2-3
2b) Listener eliminates duplicates in compound stream
Sequence Generation
Stream Splitting
Sequence Encoding
Relay
2a) Relay eliminates duplicates in the intersection of streams

Fig. 1: A brief deployment of FRER with two redundant paths

protocol enhancement. Section VI presents the related work. Lastly, in Section VII, we present a short discussion and the conclusion.

II. IEEE 802.1CB FRER

FRER is currently the only TSN standard that offers seamless protection against packet losses due to link or node failures, and malfunctioning nodes. In this section, we present a short overview of FRER and introduce the elimination problem that hinders its effective use.

A. Protocol Overview

FRER has two main mechanisms: (i) Replication of streams via different paths at the source node, i.e., TSN talker and (ii) elimination of replica packets per stream at the relay nodes or the destination node, i.e., TSN listener. In this section, we briefly discuss the enabler functions of FRER to realize those mechanisms. Fig. 1 shows the use of FRER in which three disjoint paths are assigned to a stream, e.g., two of them for redundancy. In the figure, both listener and the relay that forwards two member streams can drop the replicate packets. Generally, the talker performs

1) *Sequence Generation:* For each packet of a stream, a sequence number is generated and it is incremented after each packet in the same stream.
2) *Stream Splitting:* k copies are generated for each packet in a stream to be forwarded via k distinct paths, i.e., divided into member streams. Those paths are statically assigned using, for instance, PCR. While the packets in a member stream arrive in order, there is no guaranteed order for the packets of a compound stream, i.e., composed by all member streams across multiple paths.
3) *Sequence Encoding:* The generated sequence number is assigned to each packet, so that the same number is used for each k copies. It is stored in the Redundancy Tag (R-TAG) in the Ethernet frame. Note that R-TAG requires a sufficient size to represent the maximum sequence number for a long stream.

A listener or relay, i.e., an Ethernet switch, performs

1) *Stream Identification:* It identifies which stream the received packet belongs to so that the sequencing information associated with that stream can be used. Alternative stream identification methods such as using source or destination MAC, and VLAN ID in the Ethernet frame can be used to distinguish a stream.
2) *Sequence Decoding:* In decoding, the sequence number in the received packet is extracted to be compared to the identified stream's sequence information.
3) *Sequence Recovery:* Using the sequencing information and the decoded number, it discards or accepts the packet. If the packet is a replica due to (i) being a redundant member stream or (ii) being sent multiple times due to an erroneous sender or relay, it is dropped.
4) *Latent Error Detection:* It detects whether all expected packets received. It requires a preconfiguration to indicate the exptected number of redundant streams.

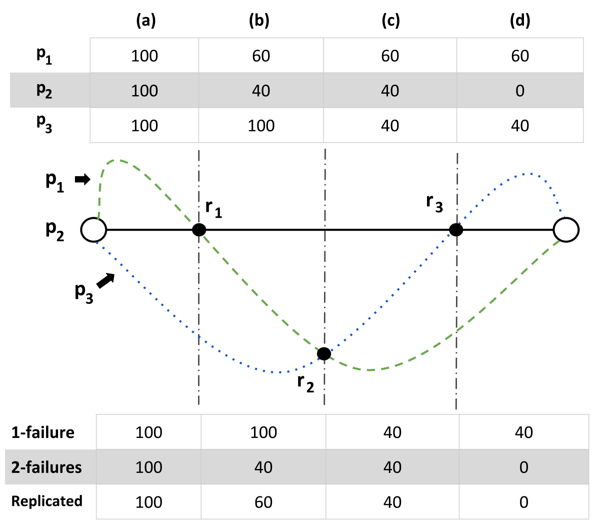
Here, the sequence recovery stage is worth further attention: (i) and (ii) correspond to sequence recovery function (SRF) and individual recovery function (IRF), respectively. While SRF processes all paths assigned to a compound stream, i.e., composed by member streams, IRF processes the stream coming from a single path, i.e., operating at a single port. A recovery function utilizes a recovery algorithm to make packet accept/forward or drop decision. The match recovery algorithm (MRA) keeps track of the received sequence numbers and drop a packet with a repeating number. The vector recovery algorithm (VRA), on the other hand, uses an acceptance interval and forwards only the packets with a sequence number in that interval to drop outdated or much ahead packets. A timeout duration is set for both algorithms to reset the expected sequence number (and interval) to refresh the recovery function in case of not forwarding any packet for the specified duration due to occasional failures.

Those functions coexist in each component with many other Ethernet protocols through the packet processing pipeline. Therefore, other intermediate functions can be performed, e.g., packet filtering rules, in-between. Further details about those functions, frame structure, and alternative stream identification methods can be found in the standard specification [4].

B. Problem Statement: Unintended Packet Elimination

In the ideal scenario, a stream is replicated on k disjoint paths, so that no relay node receives more than one copy of the same packet. Even if FRER promises a seamless integration, a network with sufficient number of disjoint paths is needed. If the network lacks such paths, the SRF may induce unexpected packet drops due to the packet elimination on **junction nodes** which lays on the intersection of multiple paths. Fig. 2 shows an edge-case scenario where a FRER-enabled talker T and listener L communicate via three partially overlapping paths, p_1, p_2, and p_3 (black, green dashed, and blue dotted lines, respectively). r_1, r_2, and r_3 represent the junction nodes, i.e, relay systems where multiple paths intersect.

The number of stream packets is investigated at the different stages between two junction nodes in the figure. While the numbers at the top show the distribution of traffic in each path by stage, the ones at the bottom show the amount of packets in the original stream protected against one or two random



	(a)	(b)	(c)	(d)
p_1	100	60	60	60
p_2	100	40	40	0
p_3	100	100	40	40

1-failure	100	100	40	40
2-failures	100	40	40	0
Replicated	100	60	40	0

Fig. 2: An edge-case scenario with three intersecting paths

failures as well as the number of packets that have replicas on redundant paths. Note that the latter is expected to be 100, as all packets should be protected by redundancy. **Stage (a).** Initially, the stream is assumed to have 100 packets replicated to p_1, p_2, and p_3, i.e., the degree of redundancy is three, in stage (a). All packets are replicated three times on this setting and one or two failures that may occur in p_1, p_2, and p_3, can be tolerated.

Stage (b). At r_1, only the first arriving packets from p_1 and p_2 will be forwarded and all replicated packets will be dropped. Note that p_1 and p_2 can carry some other streams whose amount and type, e.g., priority class, can affect which path can deliver the packets faster to r_1. Here, we assume that the total load is dynamically changing on both paths and the stream is divided into p_1 and p_2 as 60 (i) and 40 (ii) packets. Those 60 (i) packets in p_1 and 40 (ii) packets in p_2 are different, i.e., have different sequence numbers, since r_1 does not forward the same packet to the same path due to the elimination of replicas. As a result, in stage (b), in case of a failure in p_3, the main stream would be delivered as two distinct member streams via p_1 and p_3, instead of two identical streams via two disjoin paths. In case of a second failure, e.g., at p_1 and p_3, only 40 (ii) packets on p_2 would be delivered. Moreover, at maximum 60 out of 100 (i) packets are replicated on each path.

Stage (c). When the packets are forwarded faster on path p_1, r_2 drops 60 (ii) packets coming from p_3 since it has already received them from p_1 before. Note that the 40 (ii) packets sent via p_2 and p_3 are the same packets as they are remained as a result of the elimination due to the traffic in p_1. In case of the worst-case single node/link failure after r_2, i.e., p_1 fails or any 2-failures, only 40 (ii) packets can be delivered to the destination. Besides, the replicas of only 40 (ii) packets are forwarded through p_2 and p_3 whereas 60 (i) packets in p_1 are unique. Thus, in stage (c), the redundancy drops to 40 (ii) packets in contrast to the initial 100 packets.

Stage (d). As p_2 and p_3 carry the same 40 (ii) packets, r_3 eliminates the packets coming from one of those paths, say p_2. In stage (d), none of the three paths forwards all 100 packets. The overall scheme cannot tolerate 2-failures at all as the number of replica packets is virtually 0 after the ones on p_2 are eliminated.

The scenario shows that each node decreases the expected level of redundancy by eliminating replicas. For an effective use of FRER in the absence of node-disjoint paths, three important points should be considered:

- *Number of redundant paths:* In the absence of disjoint paths, it may be possible to use more than k paths to tolerate $k-1$ failures and compensate for the possible inefficiency due to the junction nodes. Therefore, it is important to evaluate the degree of redundancy to obtain the desired level of fault-tolerance.
- *Number of junction nodes:* When two paths intersect, the unintended elimination of member streams occurs as shown in the scenario. Therefore, minimizing the number of junction nodes between the selected paths is crucial.
- *Position of junction nodes:* An early intersection of two paths, e.g., within the first few hops, can affect redundancy worse as replicated packets are eliminated quickly and the stream becomes vulnerable to any failure throughout the remaining path. Therefore, this impacts the number of links ensured to be protected.

In the next section, we propose a graph metric that considers all three points to evaluate the suitability of selected paths for their expected degree of redundancy.

III. Reassurance as FRER Path Selection Metric

In this section, we introduce the novel graph metric *reassurance* that allows to select k different paths between two endpoints, so that their overlap is minimal and a potential junction node is close to the receiver. As a result, FRER-induced packet eliminations can be significantly decreased to increase fault-tolerance. We define reassurance, $\tau \in [0.0, 1.0]$ for a set of paths to quantify how close their overlapping part, i.e., junction nodes, is to the respective destination nodes if they are not disjoint. The closer to the destination, the possible packet elimination on a junction node affects shorter segments of the paths, and thus it increases the fault-tolerance against node or link failures. We then extend our definition to evaluate (i) all set of k-paths between two endpoints to find the best combination and (ii) a whole graph to check its suitability for the use of FRER by considering all combinations of endpoints and k-path in-between.

Let $G = (V, E)$ be a directed graph with nodes $u, v \in V$ and $e = (u, v) \in E$ is a directed edge from u to v s.t., $e : u \rightarrow v$. A *path* p is defined as sequence of distinct nodes connected by edges in E s.t., $p = (v_1, v_2, ..., v_n) : \exists e \in E, e : v_i \rightarrow v_{i+1})$. Accordingly, the set P_k contains the possible set of k-combinations of the paths in the graph.

As any junction node can affect the forwarded traffic on either path due to a potential packet elimination, selecting paths with the junction node(s) farthest from the origin of the path minimizes the number of links where the traffic can be affected. Therefore, we define the *longest disjoint segment* of a path, $\ell(p, C)$, that does not affect any intersection.

Definition 1. The longest disjoint segment of path p, $\ell(p, C)$, is defined as the segment of p before the first junction node between p and any other path in a particular k-combination of paths $C = \{p, p_1, p_2, ..., p_{k-1}\}$ and $C \in P_k$. Defining the overall set of junction nodes between p and $p_i \in C$ as $V_p^C = \bigcup_{p_i \in C} p \cap p_i$, the index of the first junction node on p, $t(p, C)$ is:

$$t(p, C) = \min\{i : \forall v_i \in V_p^C\} \tag{1}$$

and accordingly, $\ell(p; C)$ is

$$\ell(p, C) = \{v_i : v_i \in p \wedge i \leq t(p, C)\} \tag{2}$$

Corollary 1. The longest disjoint segment of path p is equal to p when $V_p^C = \varnothing$. Accordingly, the traffic on p is not a subject to any elimination. However, the selection of other paths in the same combination C is still important to obtain the expected degree of redundancy.

In the absence of disjoint paths, we can partially reassure the protection against a number of failures, but not all possible failures on those k paths. Accordingly, the metric *reassurance* is defined to evaluate the proportion of path segments where the flows are protected against any $k-1$ failures. Thus, the end-to-end communication is surely protected via redundancy against any failure on those segments.

Definition 2. Reassurance of a set of k paths $C \in P_k$, $\tau(C)$, is the ratio of the total length of the longest disjoint segments of all $p \in C$ to the total of number of distinct edges on those paths. When $|\ell(p, C)|$ and $|p|$ are the lengths of the longest disjoint segment of p and the whole p, respectively, $\tau(C)$ becomes

$$\tau(C) = \frac{\sum_{p \in C} |\ell(p, C)|}{\sum_{p \in C} |p|} \tag{3}$$

Reassurance enables us to evaluate a set of paths in terms of the three factors described in Section II-B. To find the best set of paths between two endpoints, i.e., the set with the highest reassurance, we should consider all possible combinations of k paths between two nodes u and v.

Definition 3. Reassurance *between two nodes* u and v, $\tau(u, v, k)$ is defined as the maximum reassurance among all k-combination of paths between u and v, $P_k^{uv} \in P_k$ s.t., $P_k^{uv} = \{p_i = (v_1, v_2, ..., v_j) : (v_1 = u \wedge v_j = v) \wedge i \leq k\}$.

$$\tau(u, v, k) = \max\{\tau(C) : C \in P_k^{uv}\} \tag{4}$$

From a practical perspective, apart from the value $\tau(u, v, k)$, it is important to obtain the best combination of paths $C \in P_k^{uv}$ to be used in the configuration of FRER. That selected combination is the one we have used in our simulations for randomly selected pairs of endpoints.

Lastly, the evaluation of the whole graph reveals to which extend it allows an effective configuration of FRER. To evaluate the whole graph, we get the average reassurance between all possible pairs of nodes in the graph.

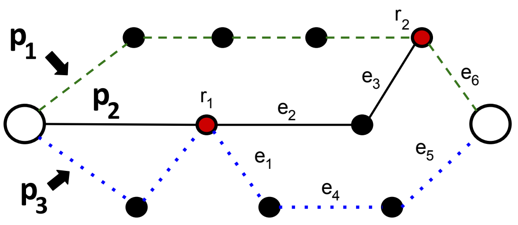


Fig. 3: A small graph to illustrate the calculation of reassurance of three paths between two nodes

Definition 4. Reassurance of a *graph* for k-redundancy is the average reassurance of all node pairs in terms of k-combinations of paths.

$$\tau(G, k) = \frac{\sum_{u,v \in V} \tau(u, v, k)}{\binom{|V|}{2}} \tag{5}$$

Corollary 2. If there are k disjoint paths between any pair of nodes in G, $\tau(G, k)$ is 1.0 and all links are protected against $k-1$ failures. In that case, a listener node is guaranteed to be received the expected amount of redundant traffic. For any value $0.0 < \tau(G, k) < 1.0$, even though some replica pakets might be dropped on the fly, the partial protection against the node and link failures is proportional to $\tau(G, k)$. Fig. 3 shows a sample graph with three paths, p_1, p_2 and p_3 between two nodes. r_1 and r_2 are the junction nodes on p_2 and p_3, and p_1 and p_2, respectively. Here, any two link failures occured after r_1 and r_2 (namely on links e_1-e_6) can disrupt the communication. For instance, when e_3 and e_6 fail at the same time and r_1 eliminates the traffic coming from p_3, no packets are received by the listener. The length of the longest disjoint segments of p_1, p_2 and p_3 are 4, 1, and 2 (i.e., until r_2 and r_1), respectively. Therefore, the reassurance is $\tau(C) = (4+1+2)/(13) = 0.54$ for $C = \{p_1, p_2, p_3\}$.

In Section V, we evaluate the effectiveness of the reassurance as a path selection strategy. We also present a brief analysis by comparing it with several graph metrics from the literature.

IV. Sequence Recovery Enhancements

Although our prior goal is utilizing a path selection strategy without modifying the protocol implementation, it is also possible to utilize SRF more effectively to avoid unintended eliminations. Note that, as discussed in Section VII, the FRER standard suggests alternative designs for the packet processing stack according to different use cases and network architectures. However, those designs may induce further complexity, where each switch utilizes different set of FRER functions and is individually configured for stream splitting and elimination. In this section, we enhance the SRF by combining it with the latent error detection function (LED) of FRER to avoid unintended eliminations without re-designing and reconfiguring the packet processing stack for each switch.

The main cause of unintended eliminations are relay systems that are not aware of whether they are junction nodes and that drop redundant packets. When a relay can infer its position as a junction node, it can forward a certain number of replicas instead of eliminating them immediately. To establish that, we made two modifications to SRF. First, a switch as part of its control plane detects by itself if it is a junction node. Second, the SRF tolerates the first k packets, where k is the junction degree, and forward rather than eliminate them. The junction degree of a switch represents the number of paths, that include the switch and are assigned to a particular stream. For a stream, if the junction degree of a switch is 0, it means that the switch is not on a path used by that stream. If it is 2, for instance, the switch expected to receive packets for that stream from two different paths. Eventually, a junction node can evaluate precisely how many replicas it should expect per packet of a stream using the junction degree and eliminate the excessive number of packets in case of, for instance, maliciously duplicated or misrouted packets after forwarding the expected replicas. Therefore, it minimizes the number of replicas to avoid unintended eliminations.

For those modifications, we assume that

1) On the control plane, we used a simplified version of PCR embracing SPB. As SPB leverages IS-IS, which is a link-state routing protocol to enable link layer routing, relays can obtain a network-wide view and thus end-to-end paths between talkers and listeners [7]. Path configuration and assignments are currently planned to be handled by PCR in the TSN standards.
2) The LED is used not only by the listener but also by intermediate relays to track the number of received replica packets per stream.

The use of IS-IS enables each relay to have a broader view of the network beyond its direct neighbors to forward packets. When utilized with a metric, for instance with *reassurance*, each relay can find end-to-end paths for a stream and calculate their junction degree. Note that as all relays have the same network view, all relay will calculate the same paths. Apart from that, if paths are configured by a controller as specified in PCR, relays could still detect how many paths they are involved in.

A relay can compute the expected number of replicas for a stream by calculating its junction degree without further configuration. It still needs to keep track of the number of received replicas to eliminate more than the expected number specified by the junction degree. Erroneous packet forwarding, malicious packet injections, or stuck senders may result in nodes receiving packets received from unexpected ingress ports. Here, the main function of LED is counting the number of packets received for a stream and alarming, for instance a controller, in those cases. Note that LED requires to be configured with the expected number of replicas (or degree of redundancy), which is computed locally as junction degree in our approach, to detect if there is any packet absent. We utilize the output of LED to forward a number of replicas limited by junction degree before elimination. Doing so, we only allow the number of replicas as many as the number of paths assigned to the stream. Any further replicas that might be received due to any error are still eliminated. Accordingly, Fig. 4 shows the main modules related to FRER and our enhancements to the packet processing pipeline of a TSN switch. From the configuration in the control plane of the switch to applying enhanced sequence recovery on the data plane, all steps are briefly shown in the figure. On the control plane of the switch, the following steps are carried out:

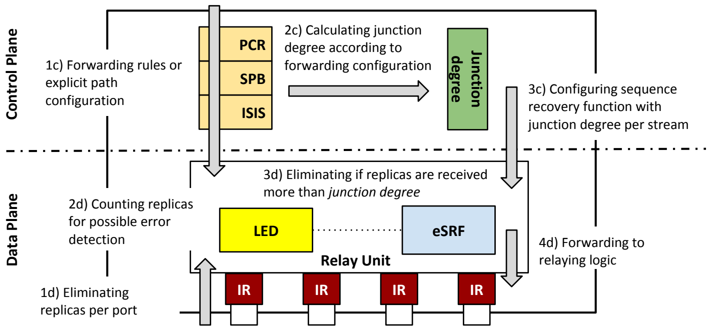

Control Plane
Data Plane
PCR
SPB
ISIS
Junction degree
1c) Forwarding rules or explicit path configuration
2c) Calculating junction degree according to forwarding configuration
3c) Configuring sequence recovery function with junction degree per stream
3d) Eliminating if replicas are received more than *junction degree*
2d) Counting replicas for possible error detection
LED
eSRF
Relay Unit
4d) Forwarding to relaying logic
1d) Eliminating replicas per port
IR
IR
IR
IR

Fig. 4: Simplified TSN switch model describing control and data plane modules

1) The relay is configured to forward the packets of stream s through given egress ports by a controller or requested to find k paths with respect to given metric, for instance maximal-disjoint or reassurance. In a distributed configuration scheme, such requests can be handeld via the Stream Reservation Protocol (SRP) [8].
2) The relay calculates the junction degree concerning the number of configured ports or calculated paths.
3) The SRF is configured with the calculated junction degree for the given stream, which is identified by the stream identification function.

When a packet is received, the following steps are carried out:

1) On each ingress port, incoming packets are identified by the stream identification function and processed by the individual recovery function, e.g., match recovery, if they have an R-TAG indicating the use of the FRER protocol. If a replica packet is received, it is directly filtered without forwarding it to the relay unit. Otherwise, packets are forwarded to the relay unit. Note that it only considers the replicas received on a single port.
2) The LED function counts the number of replicas received for stream s.
3) The **enhanced SRF (eSRF)** checks if a received packet is a replica or has a sequence number within an acceptable interval, e.g., in the acceptance window of vector recovery function. If the packet is a replica, the eSRF checks the number of received replicas for s. If that number is smaller than the relay's junction degree, the packet is not eliminated but forwarded.

4) The forwarding logic applies the remaining forwarding rules, e.g., applying any other Ethernet protocol.

V. Evaluation

In this section, we evaluate the performance of reassurance-based path selection and our enhancement for sequence recovery, eSRF. We compare reassurance-based path selection with random selection and maximum-disjoint selection, i.e., selecting the paths with the minimum number of intersections, strategies for increasing degree of redundancy (DoR), i.e., more redundant streams for protection against a higher number of possible failures. In accordance with the factors discussed in Section II-B, random, maximum-disjoint, and reassurance-based strategies represent the impact of the number of selected paths selected without any further consideration, the number of junction nodes, and the position of junction nodes, respectively. Note that apart from the position of junction nodes, reassurance-based selection also minimizes their numbers to maximize the protected segments of the paths.

After the selection of paths, we simulate $k-1$ random link failures for k redundancy. It is not possible to guarantee the desired redundancy without completely disjoint paths; thus, we aim to protect the communication against most of the failure scenarios, e.g., increasing the probability of fault-tolerance. As we examine different k, the effect of an increasing number of failures is also observed.

For the simulations, we use a very similar architecture presented in Fig. 4 without (i) the function for determining the junction degree and (ii) the eSRF for the basic, i.e., not enhanced, model. We implemented the basic and enhanced models in OMNeT++ 5.5.1v, including control plane protocols. The selected topology is analyzed offline to calculate maximum-disjoint and reassurance-based paths. Based on these precalculated paths, we simulate our FRER framework.

A. Simulation Setup

For our simulations, we used a real topology of the tier 1 network UUNET [1] as shown in Fig. 5. The network contains 49 nodes with an average node degree of 3,42. We have simulated this topology by converting UUNET nodes to TSN switches and connecting two endpoints, e.g., possible listeners and talkers, to each switch. To test higher degrees of redundancy in more connected topologies, we have also made the same conversion for randomly generated higher-connectivity topologies. For each simulation run, two nodes (connected to different switches) are randomly selected as talker and listener provided that they have a certain number of redundant paths in-between. For network traffic, we have generated a best-effort stream with 100-150 packets and uniformly-distributed interarrival time for 60 seconds.

As FRER is designed for time-sensitive systems, the selected UUNET topology as a distributed backbone network may not completely reflect their characteristics. However, due to the lack of TSN topology datasets, e.g., in-vehicle networks,

[1]The topology datasets in Table I and Fig. 5 are taken from http://www.topology-zoo.org/dataset.html

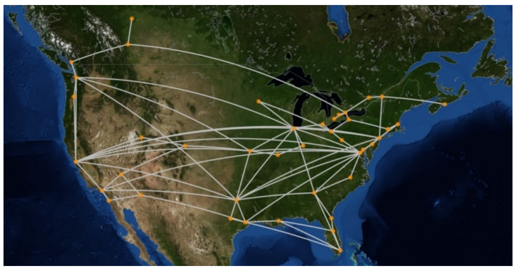


Fig. 5: UUNET in USA, 2011

we chose UUNET as a network of reasonable size and connectivity (3,42 average node degree) and thus a sufficient number of redundant paths between nodes.

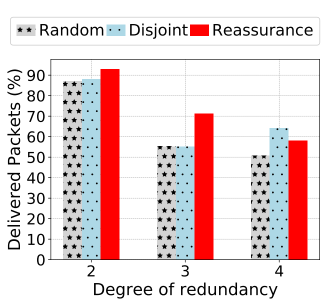
B. Results

In this section, we compare the path selection strategies in terms of packet delivery performance, path lengths and length variances as well as the position of junction nodes. Moreover, we compare reassurance with other graph metrics in the literature and show the effectiveness of our enhancement to the SRF.

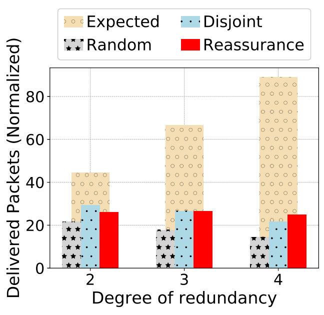
Delivery Performance. Fig. 6 shows the packet delivery ratio for each path selection strategy in dependence on the DoR, i.e., the number of redundant paths used, (a) with and (b) without simulated failures. Fig. 6a shows the results for $k-1$ random link failures and k redundancy. For $k=2$ and $k=3$, the reassurance-based selection results in the highest packet delivery ratio. Here, the expected number of packets to be delivered is the total number of distinct packets since the replica packets are not expected to be delivered in case of $k-1$ failures. For $k=4$, any combination of paths results in a large number of packet eliminations as UUNET's average node degree is less than four and thus the paths are resulting to have a lot of overlap and junction nodes. Note that no strategy, including our approach, achieves 100% packet delivery. In the absence of suitable paths in the network, the packet loss is inevitable and reassurance can only offer the best among already-intersecting FRER-paths.

Fig. 6b shows the normalized number of delivered packet for the scenarios without failures. This time, the expected number of packets (yellow bar) includes the replicas as they would also be delivered to the listener if not eliminated on the junction nodes. The figure depicts that whereas the number of replica packets is increasing with the degree of redundancy, the number of delivered packet does not significantly change because they were eliminated before being delivered to the destination. As more replicas are eliminated on the junction nodes, it results in a decreasing delivery ratio considering the increasing number of replica packets.

To test the performance reassurance-based selection in a network supporting a higher degree of redundancy than UUNET, we also generated smaller but denser random topologies with 30 to 50 nodes and a (fixed) average node degree of 4,2. Fig. 7 shows the delivery ratio in presence of $k-1$ failures



(a) With failure



(b) Without failure

Fig. 6: Delivery ratio for increasing DoR in UUNET

for a degree of redundancy of k. We observe that with an increasing number of alternative paths, the reassurance-based path selection results in a much higher delivery ratio than random and maximum-disjoint selection strategies. Even for $k = 4$, as the denser topology increases the number of disjoint paths, reassurance achieves a delivery ratio of more than 70%.

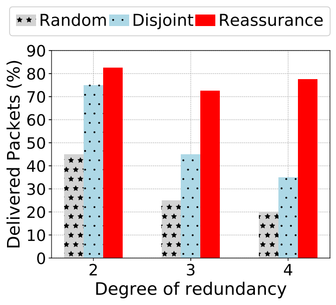
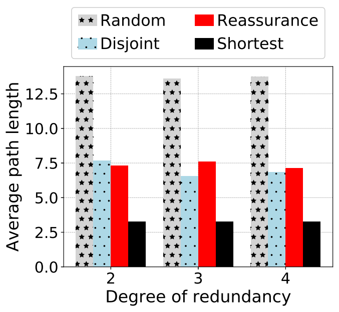


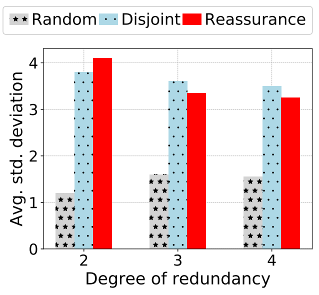
Fig. 7: Delivery ratio in a random network with failures

In case of an immediate intersection of paths, i.e., early junction nodes at first a few hops of the paths, the possibility of packet loss would be higher. The reason is that the number of links or nodes, whose failures can disrupt the service due to the unintended elimination in the early junction node, would also be higher. The reassurance-based selection gives better results since it considers the positions of those critical junction nodes and minimizes the number of those risk-induced links and nodes.

Path Lengths. The maximum-disjoint- and reassurance-based selection both take the number of junction nodes into account. Although both strategies can provide better paths for an effective use of FRER, they also have disadvantages, e.g., selecting longer paths for the sake of fewer junction nodes. Especially in TSN, longer paths require further configuration and maintainance of the traffic shapers that are decisive for the communication end-to-end latency. In fact, the maximum end-to-end latency of time-sensitive traffic classes is guaranteed up to seven hops in IEEE standards [9]. Besides, paths that differ in length significantly can lead to out-of-order packet delivery and degradation in service quality if the shorter path fails. Annex C.9 of [4] discusses that issue in more detail. Fig.



(a) Length



(b) Standard deviation in length

Fig. 8: Path length analysis for increasing DoR in UUNET

8a and Fig. 8b show the average path length and the average standard deviation in path lengths for different path selection strategies in UUNET, respectively. As depicted in Fig. 8a, while the average length of shortest paths is 2,5, both, the maximum-disjoint and the reassurance-based selection result in paths of 6,5-7,5 hops on average. Note that even though it mainly depends on the network size and characteristics, selecting redundant paths comes at the cost of an increased delay. However, both strategies still offer shorter paths than a random path selection.

Fig. 8b shows the average standard deviation in path lengths. One of the desired properties for FRER is having redundant paths that provide close end-to-end latency so that there is not a considerable jitter at the listener in case of a failure. As can be seen in the figure,, maximum-disjoint and reassurance-based selections can find paths with more than 3,5 hops deviation, which results in up to 11 hops for the average end-to-end path. Without further constraints to limit the path length, such a selection would violate the quality of service requirements of time-sensitive traffic classes that are defined for at most 7-hop paths [9]. That restriction can be satisfied by limiting the length of candidate paths in advance. Apart from such practical concerns on the standard-compliance, reassurance-based selection has no more disadvantages than the maximum-disjoint strategy.

Position of Junction Node. A junction node closer to the talker decreases the resilience against random failures due to the elimination mechanisms as discussed in Section II-B and III. The main advantage of a *reassurance-based path selection* is that paths are selected that do not overlap at all or when they overlap they do that closer to the actual destination. Fig. 9 shows the distribution of the position of junction nodes for each selection strategy in the first quarter (Q1), half, and third quarter (Q3). On the left-most histogram, most of the paths given by random selection strategy overlap on Q1, i.e., close to the talker, and thus packets get eliminated very early. As the maximum-disjoint selection does not take the position of nodes at which paths intersect into account, the paths also overlap close to the talker very often. The reassurance-based path selection results in overlapping paths closer to the listener, which decreases the early elimination of packets significantly and increases the fault-tolerance with less impact of unintended eliminations.

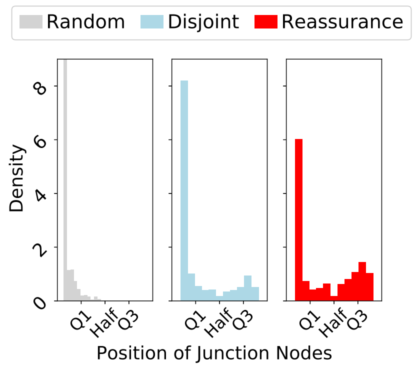


Fig. 9: Position of junction nodes in UUNET

Analysis of Reassurance. We compare reassurance and several well-known metrics to examine their possible use to evaluate the reliability of FRER in a given network. Table I shows the results of different metrics for some sample topologies to compare with the reassurance. Here, *eccentricity* of a node is the longest hop count between that node and any other node. *Betweenness* of a node is the number of shortest paths traversing that node [10]. *Closeness* of a node is the reciprocal of the sum of all shortest paths from that node to all other nodes [11]. *Assortivity* of a graph represents the correlation between the degrees of neighbor nodes to measure if nodes connect to nodes with similar degree [12]. Lastly, *clustering coefficient* of a node quantifies how close its neighbours are to being a clique [13]. All node metrics are extended to be used for the whole graph by taking average of the metric values for all member nodes. Table I also shows the correlation between selected metrics and reassurance. Here, reeassurance has a strong negative correlation (-0.97) with eccentricity and a strong positive correlation (0.99) with closeness. While eccentricity is increasing with the presence of longer paths, closeness is higher in a network with closer nodes, or shorter paths. Then, according to the correlation results, reassurance is expected to be higher in denser networks where the shortest distances between nodes are relatively shorter. Besides, those two metrics can be also used to evaluate a whole graph for an effective use of FRER. Even though eccentricity and closeness give similar results fo the whole graph, we can use reassurance to obtain the best combination of paths for end-to-end communication. In comparison to other metrics, it has a huge practical advantage for the deployment of FRER.

Performance of eSRF. As described in Section IV, we modify the SRF to provide switches a limited ability to avoid the unintended elimination of packets due to path intersections. Fig. 10 shows the results for the same scenarios in UUNET whose results are given in Fig. 6a. Under $k-1$ failures, maximum-disjoint and reassurance-based strategies offer up to 80-90% delivery ratio. Note that although their packet delivery performances are slightly different, their mostly overlapping confidence intervals show that our enhancement works equally well for both selection strategy. Besides, while all strategies are still affected by the failures, they perform better in comparison to the results in Fig. 6a. For further improvements, networks with more preferably disjoint paths are required.

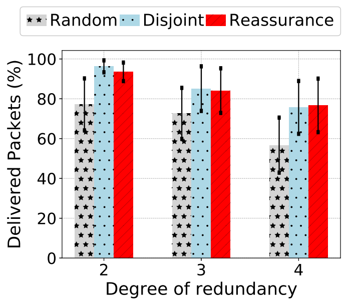


Fig. 10: Delivery ratio of enhanced sequence recovery function

Note that one of our assumptions for eSRF design is the existence of a link-state routing protocol, IS-IS, which enables path finding without a centralized controller that has a network-wide view. It may not be directly suitable for TSN that requires a tedious configuration, not only for routing but also for scheduling and time-synchronization as well. When a centralized controller exists, it can directly configure relays with their junction degrees, so that they can tolerate a certain number of duplicate packets before starting to eliminate them. The design of eSRF allows such a configuration, independent from the routing technique.

VI. Related Work

In this section, we briefly present the existing works on redundancy mechanisms in Ethernet and TSN, and other studies that analyze FRER.

Redundancy in Ethernet. The Parallel Redundancy Protocol (PRP) and the Highly-available Seamless Redundancy (HSR) are two standards that are proposed by the International Electrotechnical Commission (IEC) and provide seamless failover similar to FRER [14]. PRP offers a layer two redundancy and invisible to any higher-level network protocols. A node has to have two Ethernet interfaces with the same MAC address that are connected to two independent LANs, one for the main communication and the other one is for redundancy. Packet traffic is mirrored to both LANs to the destination nodes, which are also connected to both LANs. They process only the first arriving copy of a packet and ignore the second. In this sense, PRP provides an end-to-end redundancy, i.e., replica packets are eliminated at the end nodes.

HSR, on the other hand, is designed for networks following a ring topology without switches. A source node sends packets through opposite directions via two ports at the same time. The copies traverse the ring via both directions and the first arriving copy is processed by the destination. In this case,

TABLE I: Calculation of reassurance for $k = 3$ in various network and its correlation between well-known graph metrics

Network	Nodes	Edges	Average Degree	Reassurance.	Eccentricity	Betweenness	Closeness	Assortivity	Clustering Coefficient
Bell Canada	48	64	2.6	**0.21**	10.14	0.09	0.19	-0.22	0.15
Columbus Networks	70	85	2.4	**0.12**	13.7	0.09	0.14	-0.16	0.04
Colt Telecom	153	177	2.3	**0.04**	15.11	0.04	0.12	-0.32	0.03
UUNET	49	84	3.42	**0.41**	5.97	0.04	0.31	0.54	0.14
GTS CE	149	193	2.6	**0.07**	16.30	0.05	0.11	-0.09	0.08
Pearson correlation between reassurance					**-0.97**	**-0.06**	**0.99**	**0.75**	**0.78**

all intermediate nodes should support HSR, even if replica packets are eliminated at the end nodes.

While PRP can be easily implemented in the network stack, e.g., as a software component, HSR requires special hardware. As a result, a HSR network uses HSR nodes and modified HSR packets and requires extra interfaces to integrate standard Ethernet equipment. PRP, in contrast, can be directly deployed on Ethernet switches and uses standard Ethernet packets. In comparison to PRP and HSR, FRER is a more generic solution as it is topology-agnostic, permits communication between Ethernet devices and FRER capable ones, and thus enables seamless integration. A brief overview of those protocols and their comparison is shown in Table II.

Other Ethernet protocols such as Rapid Spanning Tree Protocol (RSTP) [15] and 802.1aq Shortest Path Bridging (SPB) [16] also offer dynamic failover. However, they do not offer real-time recovery as they switch to redundant paths only *after* detecting the failure, i.e., being reactive protocols. Moreover, there are other redundancy protocols such as Cross-network Redundancy Protocol (CRP) and Media Redundancy Protocol (MRP), which have quite similar aspects with PRP and HSR and thus omitted here.

TABLE II: Comparison between PRP, HSR, and FRER

	PRP	**HSR**	**FRER**
Implementation	Software	Hardware	Software
Frame Type	Ethernet	Custom	Ethernet
Topology	Parallel	Ring	Any
Redundancy	Active	Active	Active
Seamless integration	No	No	Yes

Redundancy in TSN. There are limited standardized reliability mechanisms offered by the IEEE TSN Task Group. Only FRER offers zero recovery time, i.e., being proactive, by redundancy [17]. In [18], the authors compare pros and cons of two different protocol stack designs for redundancy protocols: decoupled with or integreated to the stream reservation. Lastly, [19] investigates the domain-specific use of FRER focusing on real-time components in the Industry 4.0.

FRER Analysis. Existing test and simulation platforms mostly focus on time-based scheduling features of TSN protocols [20], [21]. In [22], the authors propose a simulation model for redundancy management and realize various failure scenarios to show the impact of FRER. In [23], a detailed overview of FRER is presented. Besides, the authors implement a standard-compliant model and verify the effectiveness of FRER without discussing any numerical result.

The authors of [24] present the use of FRER in various topologies and discusses practical issues and working principles. In [25], the authors discuss the limitations of FRER and specifically address buffer dimensioning for sequence numbers, inadequacy of error feedback mechanism, out-of-order delivery, extra network load, and its sensitive configuration.

Consequently, even though its dynamics, certain limitations, and coexistence with other time-based TSN protocols are discussed in a few studies, the impact of the path selection and the use of elimination mechanism are not investigated for FRER yet. In this study, we aim to address that gap.

VII. Conclusion

In conclusion, as IEEE TSN standards are very promising to become the backbone of modern safety-critical networks, FRER will be their primary reliability mechanism. For this reason, we have to ensure that it provides the expected degree of fault-tolerance. Here, we have discussed a possible misuse of the packet elimination mechanism of FRER that may cause unintended packet drops and hinder fault-tolerance.

There are alternative FRER stack designs suggested by the standards. They require further configuration of stream splitting on switches to avoid unintended packet eliminations. The functions of FRER can be flexibly positioned within the packet processing pipeline in different orders or availability, on a talker, listener, or relay nodes as described in Annex C [4] (namely Annex C). Even though such flexibility in design might be advantageous, *it is often not necessary, and if implemented naïvely, can increase the complexity [4].* Apart from flexible positioning, there are also alternative usages of internal parameters, e.g., passing *stream_handler*s through the packet processing stack to enhance the recovery logic. In this paper, we have mostly focused on the path selection to avoid possible shortfalls of the elimination mechanism regardless of different stack designs or implementation details. Fewer junction nodes results in a decreased probability that a single failure can affect multiple paths at the same time. For that, we proposed the novel metric, reassurance, to evaluate the suitability of paths to be used for FRER. Our simulation results indicate that *reassurance* can be utilized in an effective path selection strategy. It is less affected by random node failures in comparison to a random and maximum-disjoint path selection.

Moreover, we have introduced an enhancement on the sequence recovery function (eSRF) independent and show that it overcomes the unintended elimination of packet.

References

[1] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-Vehicle Networks: A Review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, pp. 534–545, April 2015.

[2] S. Schneele and F. Geyer, "Comparison of IEEE AVB and AFDX," in *IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, pp. 7A1–1–7A1–9, Oct 2012.

[3] "Time-Sensitive Networking (TSN) Task Group." Available at https://1.ieee802.org/tsn/.

[4] "IEEE Standard for Local and Metropolitan Area Networks–Frame Replication and Elimination for Reliability," *IEEE Std 802.1CB-2017*, pp. 1–102, Oct 2017.

[5] "IEEE Standard for Local and Metropolitan Area Networks Bridges and Bridged Networks Amendment 24: Path Control and Reservation," *IEEE Std 802.1Qca-2015*, pp. 1–120, March 2016.

[6] "IEEE 802.1aq Shortest Path Bridging (SPB)." Available at https://1.ieee802.org/tsn/802-1aq-shortest-path-bridging/.

[7] D. Fedyk, D. Allan, P. Ashwood-Smith, N. Bragg, J. Farkas, M. Ouellete, M. Seaman, and P. Unbehagen, "RFC 6329: IS-IS Extensions Supporting IEEE 802.1aq Shortest Path Bridging," tech. rep., Internet Engineering Task Force, 2012.

[8] "IEEE Standard for Local and Metropolitan Area Networks–Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)," *IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005)*, pp. 1–119, 2010.

[9] "IEEE Standard for Local and Metropolitan Area Networks–Audio Video Bridging (AVB) Systems," *IEEE Std 802.1BA-2011*, pp. 1–45, 2011.

[10] U. Brandes, "On Variants of Shortest-Path Betweenness Centrality and their Generic Computation," *Social Networks*, vol. 30, no. 2, 2008.

[11] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215 – 239, 1978.

[12] M. E. J. Newman, "Mixing patterns in networks," *Phys. Rev. E*, vol. 67, p. 026126, Feb 2003.

[13] T. Schank and D. Wagner, *Approximating clustering-coefficient and transitivity*, vol. 2004 of *Interner Bericht. Fakultät für Informatik, Universität Karlsruhe*. Universität Karlsruhe, Karlsruhe, 2004.

[14] "Industrial communication networks - High availability automation networks - Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR)," *IEC 62439-3:2016 RLV*, pp. 1–540, March 2016.

[15] "IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges," *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pp. 1–281, June 2004.

[16] M. Seaman and D. Fedyk, "IEEE Std 802.1aq Shortest Path Bridging," tech. rep., IEEE 802.1 Working Group, 2012.

[17] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research," *IEEE Communications Surveys Tutorials*, vol. 21, pp. 88–145, Firstquarter 2019.

[18] S. Kehrer, O. Kleineberg, and D. Heffernan, "A comparison of fault-tolerance concepts for IEEE 802.1 Time Sensitive Networks (TSN)," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pp. 1–8, Sep. 2014.

[19] F. Prinz, M. Schoeffler, A. Lechler, and A. Verl, "End-to-end Redundancy between Real-time I4.0 Components based on Time-Sensitive Networking," in *IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 1083–1086, Sep. 2018.

[20] J. Jiang, Y. Li, S. H. Hong, A. Xu, and K. Wang, "A Time-sensitive Networking (TSN) Simulation Model Based on OMNET++," in *IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 643–648, Aug 2018.

[21] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, and K. Rothermel, "NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++," in *2019 International Conference on Networked Systems (NetSys)*, pp. 1–8, IEEE, 2019.

[22] M. Pahlevan and R. Obermaisser, "Redundancy Management for Safety-Critical Applications with Time Sensitive Networking," in *28th International Telecommunication Networks and Applications Conference (ITNAC)*, pp. 1–7, Nov 2018.

[23] S. Qian, F. Luo, and J. Xu, "An Analysis of Frame Replication and Elimination for Time-Sensitive Networking," in *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, ICNCC 2017, (New York, NY, USA), p. 166–170, Association for Computing Machinery, 2017.

[24] G. Ditzel, "High Availability in EtherNet/IP Systems Using Frame Replication and Elimination for Reliability (FRER) as defined in the TSN Standard IEEE 802.1CB-2017," in *ODVA 18th Industry Conference and Annual Meeting*, October 2018.

[25] R. Hofmann, B. Nikolić, and R. Ernst, "Challenges and Limitations of IEEE 802.1CB-2017," *IEEE Embedded Systems Letters*, pp. 1–1, 2019.

Appendix H

Implementation and Orchestration of IEEE 802.1CB FRER in OMNeT++

Abstract

IEEE 802.1 Time-Sensitive Networking (TSN) family of standards enables real-time and deterministic networks on top of standard Ethernet. It also offers a seamless redundancy protocol, IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER), to protect the network against node and link failures. Although its effective use is currently being studied by many researchers, the management of FRER can be complicated and limited to only small topologies. Manual configuration of switches is required as there is not any implemented management protocol yet. In this paper, we describe our implementation of the FRER protocol in OMNeT++. Besides, we implement additional control plane protocols, Intermediate System to Intermediate System (IS-IS) and IEEE 802.1aq Shortest Path Bridging (SPB), for network discovery and link layer routing to configure FRER properly for multi-path communication. As SPB is the main instrument of the path reservation mechanism in TSN, our framework enables to simulate large-scale time-sensitive networks together with reliability and orchestration mechanisms. The whole implementation is available as open-source.

Reference

Doğanalp Ergenç, M. Fischer. Implementation and Orchestration of IEEE 802.1CB FRER in OMNeT++. IEEE International Conference on Communications (ICC), Workshop on Time-sensitive and Deterministic Networking, 2021.

Contribution

In the forementioned publication, the whole contribution belongs to this thesis. The co-author helped to improve the quality of the paper with his valuable feedback.

Implementation and Orchestration of IEEE 802.1CB FRER in OMNeT++

Doğanalp Ergenç, Mathias Fischer
University of Hamburg
{ergenc, mfischer}@informatik.uni-hamburg.de

***Abstract*—IEEE Time-sensitive Networking (TSN) family of standards enables real-time and deterministic networks on top of standard Ethernet. It also offers a seamless redundancy protocol, 802.1CB Frame Replication and Elimination for Reliability (FRER), to protect the network against node and link failures. Although its effective use is currently being studied by many researchers, the management of FRER can be complicated and limited to only small topologies. Manual configuration of switches is required as there is not any implemented management protocol yet. In this paper, we describe our implementation of the FRER protocol in OMNeT++. Besides, we implement additional control plane protocols, Intermediate System to Intermediate System (IS-IS) and 802.1aq Shortest Path Bridging (SPB), for network discovery and link layer routing to configure FRER properly for multipath communication. As SPB is the main instrument of the path reservation mechanism in TSN, our framework enables to simulate large-scale time-sensitive networks together with reliability and orchestration mechanisms. The whole implementation is available as open-source.**

***Index Terms*—TSN, FRER, SPB, IS-IS, simulation, OMNeT++**

I. Introduction

The family of Time-sensitive Networking (TSN) standards have been recently proposed by the IEEE 802.1 Working Group. They extend IEEE 802.3 Ethernet standard [1] to satisfy time-sensitive communication requirements of mission-critical systems, e.g., in automotive, aviation, and industrial networks. TSN offers a set of protocols, which can be implemented on commodity-off-the-shelf Ethernet hardware, to manage different traffic classes, ensure deterministic communication within a bounded delay, define filtering and networking policies, and improve reliability by seamless redundancy.

802.1CB Frame Replication and Elimination for Reliability (FRER) is the primary solution for TSN to tolerate link and node failures [2]. For that, it offers a static redundancy mechanism by replicating each packet, i.e., Ethernet frame, via multiple redundant paths. FRER also comes with an elimination mechanism that runs on relay systems and end-hosts to drop replica packets. This mechanism drops replica packets and protects the network against loops and babbling idiots.

Although FRER introduces a set of functions to implement reliable communication, it still requires a tedious configuration scheme to utilize those functions. In the standard, alternative deployments and configurations are specified. For example, while it is possible to assign multiple static redundant paths between a talker and listener, configuring each intermediate switch on an end-to-end path to initiate packet duplications on the fly is also an alternative that provides further flexibility in spite of requiring manual configuration. Therefore, we have to deploy a controlling mechanism, preferably following the (control plane) architectures in the respective standard TSN [3], to find multi-hop end-to-end paths and utilize FRER effectively.

Currently, there are only a limited number of providers implementing TSN protocols in their equipment. Besides, as TSN has not been broadly deployed in the real systems yet, researchers mostly rely on the simulation tools and frameworks to evaluate the protocols. Accordingly, in this study, we implement the FRER framework, including a controlling scheme in one of the most popular network simulators, OMNeT++ [4]. The main contributions of our paper are:

- We extended the open-source and actively developed OMNeT++ framework CoRE4INET [5] by our FRER implementation.
- We described our implementation of the link layer routing protocol Intermediate System to Intermediate System (IS-IS) and a simplified version of the network management protocol 802.1aq Shortest Path Bridging (SPB) [6] to configure FRER. Note that SPB is the protocol used as a basis for the path configuration mechanism of TSN as specified in the standard 802.1Qca Path Control and Reservation (PCR) [3].
- We described additional FRER configuration extensions to enable assigning static end-to-end paths in the simulation environment.

The rest of the paper is organized as follows. Section II introduces the related work on FRER analysis and implementation and the control plane of TSN. Section III briefly explains FRER and related control plane protocols. Section IV summarizes the implementation details of our framework as well as the remaining features. Section V concludes the paper.

II. Related Work

Several studies analyze different aspects of FRER [7]–[9] but only a few provides a practical simulation tool or framework. In [10], the authors propose a simulation model for temporal properties and redundancy management, e.g., FRER, in TSN networks and realize various failure scenarios to show the impact of FRER. Their implementation is in Riverbed (formerly OPNET) and it is not an open-source framework.

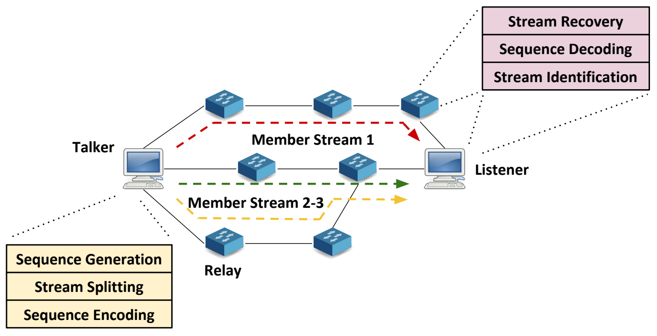


Fig. 1. An example deployment of FRER

In [11], a detailed overview of FRER is presented. Besides, the authors implement a standard-compliant model and verify the effectiveness of the features of FRER without discussing any numerical result. However, it is not possible to use it with other existing TSN simulation frameworks. Lastly, [12] proposes another FRER implementation for an older version of OMNeT++. It enables to test the functions of FRER only in small setups as it does not have any control plane routing and configuration protocol.

There are only some limited works focusing on the control plane protocols addressing IS-IS and SPB. [13] proposes an SPB simulation framework for the network simulator NS-3. In [14], the authors implement IS-IS in a custom testbed, which is now outdated. However, there is not a single platform to use FRER and the control plane protocols together. Accordingly, we implemented them holistically in OMNeT++ utilizing the existing TSN framework CoRE4INET.

III. Background

In this section, we briefly introduce FRER and alternative controlling mechanisms to configure FRER.

A. IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER)

FRER has two main mechanisms: (i) Replication of streams through different paths (or links) in the source node, i.e., TSN talker and (ii) elimination of replica packets per stream in the relay nodes or the destination node, i.e., TSN listener. In this section, we briefly discuss the enabler functions of FRER to realize those mechanisms.

Fig. 1 shows the use of FRER in which three disjoint paths are assigned to a stream, e.g., two of them for redundancy. In the figure, both listener and the relay that forwards two member streams can drop the replicate packets. Generally, the talker performs (i) *sequence generation* by generating a unique identifier per packet of a stream to be incremented for the other packets in the sequence, (ii) *stream splitting* by copying the packets and creating member streams to be sent through k distinct paths, and (iii) *sequence encoding* by assigning a sequence number to the copied packets via the Redundancy Tag (R-TAG).

A listener or relay, i.e., an Ethernet switch, performs (i) *stream identification* by applying a stream identification function (e.g., a function taking destination MAC and VLAN ID of a packet as input) to distinguish a stream, (ii) *sequence decoding* by extracting the sequence identifier of a packet to be compared to the identified stream's sequence information (iii) *stream recovery* by deciding if a packet is duplicate and should be dropped or forwarded, and lastly (iv) *latent error detection* by counting if it has received the expected number of duplicate packets to detect a node or link failure on the path of a member stream.

Here, the stream recovery stage consists of two functions. The sequence recovery function (SRF) processes all the packets received from different ports of the switch, and thus, it can detect the duplicate packets of a stream coming from different paths. The individual recovery function (IRF), on the other hand, processes the stream coming from a single path (or port) and is effective against, for instance, duplicate packets due to a stuck sender. Any recovery function utilizes a recovery algorithm to make packet accept/forward or drop decision. Match recovery algorithm (MRA) keeps track of the received sequence numbers and drop a packet with a repeating number. Vector recovery algorithm (VRA), on the other hand, uses an acceptance interval and forwards only the packets with a sequence number in that interval to drop outdated or much ahead packets. A timeout duration is set for both algorithms to reset the expected sequence number (and interval) to refresh the recovery function in case of not forwarding any packet for the specified duration due to occasional failures.

B. Intermediate System to Intermediate System (IS-IS)

IS-IS is a link-state routing protocol, operating by flooding topological information through the network so that each bridge can have a network-wide topological view. It supports both link and network layer routing and enables routers or bridges to build a database of network topology [15]. IS-IS utilizes Dijkstra's shortest path algorithm to find end-to-end paths between network elements.

IS-IS assumes that the network is divided into non-intersecting areas. Inside an area, Level 1 (intra-area) bridges exchange information to update their link state databases. For inter-area communication, Level 2 bridges are deployed in the backbone of the network. Lastly, Level 1-2 bridges take a gateway role connecting Level 1 and Level 2 bridges in different areas. This area-division helps to create isolated domains, which are easier to configure and manage by different tenants.

Each bridge periodically sends *HELLO* packets to establish adjacency between neighbor bridges. Apart from that, they periodically send their databases to maintain the topological view and detect any inconsistency in-between. As a result of that consistency-check, a bridge can request a partial database to confirm the differences and update if necessary. Lastly, the communication scheme of IS-IS allows to the definition of custom packets by extending the existing ones with extra type-length-values (TLVs) to modify the protocols.

C. IEEE 802.1aq Shortest Path Bridging (SPB)

IEEE 802.1aq Shortest Path Bridging (SPB) is a native Ethernet solution that offers a significant level of isolation through virtual networks, improves drawbacks of Spanning Tree Protocol (STP), and has strong features for traffic engineering in link layer [6], [16]. SPB extends the use of logical networks defining Service Identifiers (ISIDs) beyond standard virtual local area networks (VLANs). It enables to create more extensive broadcast domains. In Backbone Edge Bridges (BEB), i.e., bridges that end hosts are directly connected to, ISIDs are configured and matched with VLAN IDs that hosts/ports are registered. To perform end-to-end routing, BEBs discover all other bridges, e.g., other BEBs and also backbone core bridges (BCBs) that are not connected to any host nodes but connect other switches and corresponding ISIDs they serve.

For network discovery, SPB deploys IS-IS, whose details are briefly explained in Section III-B, as its control plane to obtain a network-wide view. SPB then enables the assignment of the shortest paths to streams, and also supports multipath configuration for the best QoS. For the shortest path selection, SPB utilizes 16 different equal-cost path tie-breaking (ECT) algorithms so that when all bridges are configured to use the same ECT, they can select the same shortest paths without any further configuration by, for example, a centralized controller. As a result, ECTs provide congruency and symmetry for the path selection.

SPB comes with two modes, SPBV and SPBM, using (i) seamless address and VLAN translation and (ii) MAC-in-MAC encapsulation respectively to manage extended logical networks. SPBV is designed for smaller enterprise networks and enables data plane MAC learning in accordance to the plug-and-play nature of Ethernet. In contrast, SPBM leverages MAC learning in the control plane, manages those addresses, and thus enables the orchestration of larger-scale networks with comprehensive virtual networking features.

Lastly, SPB is utilized by 802.1Qca Path Control and Reservation (PCR) as the path configuration mechanism for TSN [3]. Therefore, it is an important protocol for the management of static paths in TSN-enabled systems.

IV. Implementation

In this section, after giving an overview of the main modules of our implementation, we describe each in more detail in the following sections. Besides, we list the remaining features that are not included and left as future work.

We implemented FRER and the control plane protocols on top of OMNeT++ v5.5.1; and the frameworks INET v3.6.7 and CoRE4INET [5] with the latest version by June 2020 supporting given OMNeT++ and INET versions. CoRE4INET is an extension to the INET framework for real-time Ethernet simulation. It includes several TSN standards, including filtering, traffic shaping, and time synchronization protocols, as well as older Audio Video Bridging (AVB) standards. Here, we extend CoRE4INET with the redundancy protocol FRER and the control plane protocols inheriting some of its modules and

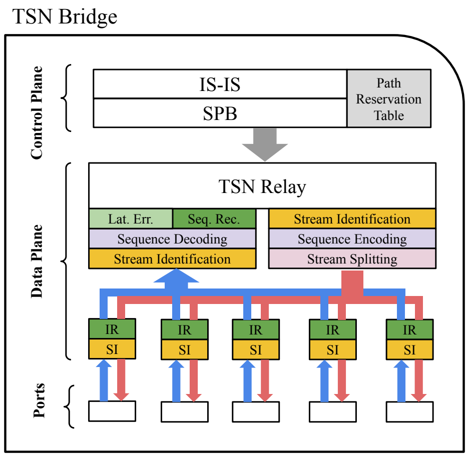


Fig. 2. Overall bridge architecture containing all modules. FRER functions are shown in different colors. The functions with the same color are the same modules utilized for different stages of packet processing.

nodes. More details on the extensions and additional features are given in the rest of this section.

A. General Overview

We have implemented five main modules, as shown in Table I with respect to the extended OMNeT++ modules and their types. Note that *type* represents if respective module is designed as network description files (NED), implemented in C++ and have a source code (code), only configured as a parameters in another module (parameter), or a message file (message).

TSNBridge is the main component that contains all other modules and defines their relationships, e.g., connected via input/output ports to or direct access among each other, and it is an extended version of *TSNEtherSwitch* defined by CoRE4INET. *TSNRelay* extends *SRPRelay* by reimplementing packet forwarding and adding FRER functions and control plane protocols. *SPB* and *ISIS* are the control plane protocols for link layer routing and path assignment. Lastly, *FRER* is not implemented in a single module but as individual functions to be distributed to the different stages of packet processing.

Fig. 2 shows the overall bridge architecture, which represents *TSNBridge*. It illustrates inter-module relationships in control and data plane as well. In the following sections, we present the details of those components under different categories.

B. Bridge and Relay

TSNBridge utilizes all FRER functions as shown in Fig. 2. Ingress and egress ports are suited with different functions, which do not have to be actively used in each bridge. For instance, while edge bridges may require sequence encoding and splitting, core bridges can directly forward received packets after stream identification. By default, we have deployed all functions as illustrated. Note that, from an implementation perspective, *TSNBridge* has not been programmed, i.e.,

TABLE I
DESCRIPTION OF THE MAIN MODULES

Module	Extension (of)	Type	Description
TSNBridge	TSNEtherSwich (CoRE4INET)	NED and code	Contains all other modules and define intermodule relationships
TSNRelay	SRPRelay (CoRE4INET)	NED and code	Defines packet forwarding logic instrumenting other modules
FRER	N/A	NED, code, message	A set of functions implementing FRER features
SPB	cSimpleModule	NED, code, message	Implements paths allocation utilizing IS-IS and path reservation table
ISIS	cSimpleModule	NED, code, message	Implements link layer routing and network discovery

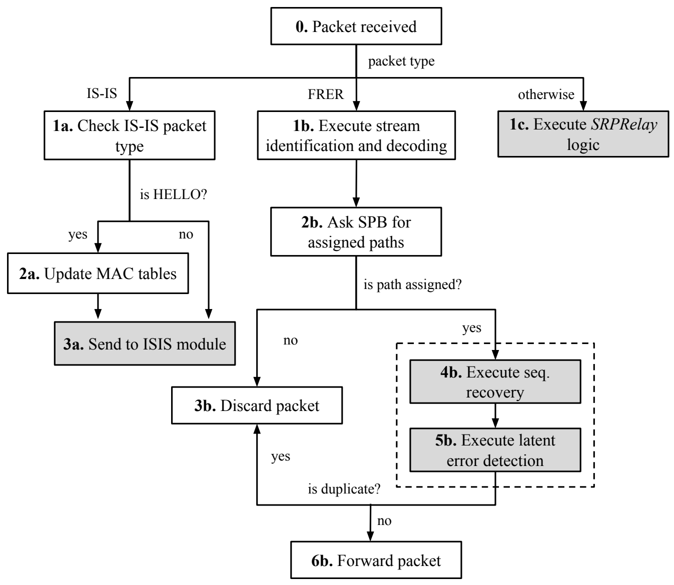

0. Packet received
packet type
IS-IS
FRER
otherwise
1a. Check IS-IS packet type
1b. Execute stream identification and decoding
1c. Execute *SRPRelay* logic
is HELLO?
yes
no
2a. Update MAC tables
3a. Send to ISIS module
2b. Ask SPB for assigned paths
is path assigned?
no
yes
4b. Execute seq. recovery
5b. Execute latent error detection
3b. Discard packet
yes
is duplicate?
no
6b. Forward packet

Fig. 3. Processing received packets in *TSNRelay*. Grey boxes show the external modules (implemented or inherited) used by the relay.

does not have source code, but it only defines inter-module relationships, i.e., by network description (NED) modules in OMNeT++.

Apart from FRER functions, *TSNRelay* and control plane protocols are deployed in *TSNBridge*. The relay module orchestrates the packet processing stages such as where and when to apply FRER functions on frames and how to access and update routing information by employing other modules. It extends existing *SRPRelay* module to support not only SRP packets but IS-IS and FRER frames as well. Fig. 3 shows a flow chart that represents the overall logic of the relay, mainly implemented as a single packet handling function, i.e., *handleMessage(cMessage)* in the source code. According to its type, a frame is processed either by *ISIS* module (1a-3a), FRER functions (1b-6b), or inherited *SRPRelay* (1c). IS-IS control packets are sent to the *ISIS* module. Before that only *HELLO* packets are also used to update the MAC table of the relay for neighborhood discovery. FRER frames, on the other hand, are processed by FRER functions and forwarded according to the paths assigned by SPB. For the rest of the frames, the relay inherits the behavior of *SRPRelay*, which is provided by CoRE4INET.

Note that, before a frame received by *TSNRelay*, it is already processed by an individual recovery function. It represented as separated from the relay module in Fig. 2 and thus is not shown in Fig. 3. Besides, in the steps shown as grey rectangles, the frames are sent to other modules, which are connected internally under *TSNBridge*. Their details are given in the following sections.

C. Data Plane

In the data plane, we have implemented stream encoding/decoding, sequence recovery, and latent error detection functions of FRER. Table II shows the implemented FRER functions, corresponding OMNeT++ modules and their types, and relevant parameters to configure the modules.

TABLE II
FRER FUNCTIONS, MODULES AND THE RELEVANT DATA PLANE PARAMETERS

Module	Type	Parameters
Stream Identification	NED	*sidFunction*
SequenceGeneration	Code	N/A
SequenceEncoder	NED and code	N/A
Stream splitting	Parameter	*splitFactor*
StreamRecovery	NED and code	*recoveryType*
		streamRecoveryFunction
LatentErrorDetection	NED and code	*splitFactor*
FRERbusEthernetFrame	Message	*sequenceNumber*

Outgoing frames are first sent to *StreamIdentification* function to obtain its identifier for the respective stream. Then, *SequenceEncoder* module, where the configured stream identification function is executed queries *streamSequenceTable* to obtain the latest observed sequence number for that stream using the identifier. If it is not observed before, i.e., the first frame of the stream, a random sequence number is generated via *SequenceGeneration* function, and the frame is encoded accordingly. Otherwise, the existing number is incremented and encoded to the frame.

Note that stream identification and generation functions are not defined as distinct OMNeT++ modules but a set of static functions in *StreamIdentification* and *SequenceGeneration* modules respectively to be used by all other modules without requiring extra intra-module connections. Stream identification is designed as a single function getting the frame and the function type, *sidFunction*, as parameters, as FRER standard enable the use of multiple identification functions. Currently, (i) null identifier, (ii) source address and VLAN, and (iii) destination address and VLAN are available in the framework [2].

Lastly, encoded frames are splitting through the ports that are configured by the control plane protocols. Stream splitting is not desgined as an independent module but a configuration parameter of *TSNRelay*, *splitFactor*, that specifies the number of distinct paths that the duplicate frames are sent. The configuration of those paths are explained in Section IV-D.

Incoming frames are processed as shown in Fig. 3. On ingress ports, two types of sequence recovery function take place: Individual recovery and sequence recovery. In the implementation, *StreamRecovery* module can be configured as either function via *recoveryType* parameter. After the stream identification, the individual recovery function eliminates duplicates received on a particular port. For the elimination, match or vector recovery functions can be used according to the configuration of *StreamRecovery* module via the parameter *streamRecoveryFunction*. Then, the frames are sent to the stream recovery function, which uses stream identification functions and latent error detection module, implemented in *LatentErrorDetection*. This module detects missing or injected frames, i.e., fewer or more duplicates than the expected number. To implement further alarming schemes, this module should also be configured the same value with *splitFactor*.

As stream recovery functions can be placed in both relays and end-host, the rest of the process is node-dependent. After the sequence recovery process in a relay, the frame is processed by the relay module and reinjected to the packet processing loop to be forwarded, if not dropped. On the other hand, the end-host makes the final decision to either accept or drop the frame, e.g., checking if it is the destination nodes or the frame is a duplicate, and thus data plane processing is completed at that point.

Lastly, apart from the FRER functions, It is required to define an extended Ethernet frame including R-TAG, which carries the sequence number and *EtherType* representing FRER frames. For this, we implement *FREREthernetFrame* extending existing IEEE 802.1Q frames. The type of those frames, i.e., in EtherType field, is hardcoded as 0xF1C1.

D. Control Plane

The control plane consists of IS-IS and SPB. Accordingly, we have implemented NED modules, source codes, and custom messages for both modules. Table III summarizes the implemented modules including their configuration parameters.

TABLE III
CONTROL PLANE MODULES

Module	Type	Parameters
ISIS	NED, code, message	*bridgeID*
		hostID
		updatePeriod
		dbAnnouncePeriod
SPB	NED and code	*isid*
		paths
PathSelectionTable	NED and code	*timeout*

IS-IS works as the main routing protocol realizing network discovery, topology information sharing and verification, and routing, as explained in Section III-B. In the implementation, each bridge has a unique identifier, *bridgeID*, that can be used instead of MAC addresses to construct paths without keeping track of all MAC addresses for an easier representation. Note that *bridgeID*-MAC address matchings are indeed shared between the bridges during the discovery. This identifier is incremented automatically for each bridge during the initialization. The configuration parameters of IS-IS module are *updatePeriod* and *dbAnnouncePeriod*, which specify the periods for sending *HELLO* and *DB_ANNOUNCE* packets, respectively. *HELLO* packets are used for the discovery of immediate neighbors and each bridge sends its *isis* database via *DB_ANNOUNCE* packets. For broadcasting, the existing IS-IS multicast address in OMNeT++ is used. That address is also used to distinguish received *isis* packets on the relay as well as the IS-IS Ethernet tag, which is 0x88cc. We have implemented a new *ISISPacket* extending *EthernetIIFrame* to embed the required meta data, e.g., *bridgeID*, VLAN IDs, link cost etc.

When a *HELLO* packet is received, the IS-IS module of the bridge records the source of the packet as its direct neighbor in its database. Similarly, after a database announcement packet is received, the module updates its IS-IS database. Lastly, the module provides Dijkstra's shortest path function to find end-to-end paths between given source and destination MAC addresses as well as the source and destination *bridgeID* in case of inter-bridge routing.

For a simpler and more extensive network discovery, we have also defined an end-host IS-IS agent that enables end-hosts to announce their MAC addresses to edge bridges. While using the same mechanism described above, only the database announcement process is excluded as path finding is irrelevant to end-hosts. Instead of a *bridgeID*, an end-host is represented by a unique *hostID*

Utilizing the IS-IS module, **SPB** module is responsible to assign multiple paths to a stream between given end-hosts (or practically between given edge bridges). SPB has *isid* parameter representing the Service Identifier (ISID) that helps to define extended virtualized domains together with VLAN IDs. Accordingly, we have modified the path-finding function of IS-IS module to construct the paths only among the bridges having the same *isid* so that we can define intra-domain paths. However, it also requires to disseminate ISIDs so that a bridge can know with which other bridges it is in the same service group. ISID is accordingly embedded to the IS-IS *HELLO* packets. Note that as IS-IS enables to define custom packet fields via TLVs, such an embedding is easy in practice.

In contrast to the IS-IS module, SPB does not have any proactive mechanisms to be maintained periodically. Instead, as shown in Fig. 3 3b., it responds with the respective paths that are assigned to a stream when the relay module requests. The assignment of paths can be performed in alternative ways. SPB can directly find k shortest paths s.t. $k = splitFactor$ to satisfy the redundancy requirements of FRER. In our implementation, as *TSNRelay* has $splitFactor$ as a configuration parameter, it can directly request the required number of paths to split a stream. We have also implemented k shortest-disjoint path finding for the intended use of FRER. The tie-breaking algorithms of SPB helps to consistently select the same k paths for each bridge locally. Another approach is the manual configuration that a network manager can directly provide paths specified by a sequence of *bridgeID*s. For instance, a centralized controller directly configures SPB module to respond with the given paths for the respective streams. We have defined a parameter *paths* that takes semicolon-separated paths. In this format, each path is represented as $T : R_1, R_2, ... : L$, where T and L are the *bridgeID* of the edge bridges that talker and listener are connected to. Similarly, R_i is the identifier of any intermediate relays along the path.

In both assignment approaches, SPB records the paths in *PathReservationTable* that enables to store multiple paths to be used for the incoming packets of the same stream within a specified timeout duration. It is an extended MAC table that stores end-to-end paths instead of only the next hops. However, the default MAC table also exists to forward non-FRER traffic. The timeout is configured by the parameter *timeout*.

E. Remaining Features

We highlight some of the remaining features to encourage the readers to contribute and behold when using the the framework.

We have implemented most of the features of FRER in a modular fashion. Some features are simplified to reduce the complexity of the framework. Those simplications are:

- The sequence generation function does not keep track of the generated numbers, which is required to ensure that no number is repeated within a certain period.
- Only the stream identification functions utilizing layer 2 packets headers are implemented. IP headers can also be included as given in the standard.
- The *splitFactor* parameter is set for each node, not for each stream. Therefore, all FRER-tagged streams have the same level of redundancy.

We list some of the simplified features of IS-IS as:

- The area-division logic has not been implemented and all bridges are assumed to be Level 1.
- A bridge updates its database with the most recent topology announcement packet instead of requesting partial databases in case of inconsistency.

Similar, SPB has also some limitations, which are:

- Our implementation includes the features of both SPBV and SPBM without a strict distinction.
- Only two ECTs are implemented. There might be a need for more to select a higher number of shortest paths for an increased redundancy for FRER.
- The manual configuration only lets the injection of pre-configured paths, whether before initialization or at a specific simulation time.

V. Conclusion

In this paper, we present our implementation of 802.1CB Frame Replication and Elimination for Reliability (FRER), which is the only reliability mechanism for IEEE Time-sensitive Networking (TSN), and the required management protocols for an effective use of FRER. We have extended an existing TSN framework in the network simulator OMNeT++ with FRER, IS-IS, and SPB to enable multipath redundancy, network discovery, and routing in larger-scale time-sensitive networks. As IS-IS and SPB are the main protocols that are used to implement the path reservation mechanism in TSN, those modules are important to have a more extensive TSN simulation environment. A more detailed configuration and respective numerical results obtained by the use of this framework can be found in our predecessor study [9] that we analyze the reliable use of FRER. Lastly, the framework is published as open-source at *https://github.com/UHH-ISS/omnet-802.1cb*.

References

[1] "Time-Sensitive Networking (TSN) Task Group." Available at https://1.ieee802.org/tsn/.

[2] "IEEE Standard for Local and Metropolitan Area Networks–Frame Replication and Elimination for Reliability," *IEEE Std 802.1CB-2017*, pp. 1–102, Oct 2017.

[3] "IEEE Standard for Local and Metropolitan Area Networks Bridges and Bridged Networks Amendment 24: Path Control and Reservation," *IEEE Std 802.1Qca-2015*, pp. 1–120, March 2016.

[4] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *Proc. of the 1st Int. Conf. on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, 2008.

[5] T. Steinbach, H. Dieumo Kenfack, F. Korf, and T. C. Schmidt, "An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy," in *Proc. of the 4th Int. Conf. on Simulation Tools and Techniques*, SIMUTools '11, 2011.

[6] "IEEE 802.1aq Shortest Path Bridging (SPB)." Available at https://1.ieee802.org/tsn/802-1aq-shortest-path-bridging/.

[7] F. Prinz, M. Schoeffler, A. Lechler, and A. Verl, "End-to-end Redundancy between Real-time I4.0 Components based on Time-Sensitive Networking," in *IEEE 23rd Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2018.

[8] R. Hofmann, B. Nikolić, and R. Ernst, "Challenges and Limitations of IEEE 802.1CB-2017," *IEEE Embedded Systems Letters*, 2019.

[9] D. Ergenc and M. Fischer, "On the Reliability of IEEE 802.1CB FRER," in *IEEE Int. Conf. on Computer Communications (INFOCOM)*, 2021.

[10] M. Pahlevan and R. Obermaisser, "Redundancy Management for Safety-Critical Applications with Time Sensitive Networking," in *28th Int. Telecom. Networks and Applications Conf. (ITNAC)*, Nov 2018.

[11] S. Qian, F. Luo, and J. Xu, "An Analysis of Frame Replication and Elimination for Time-Sensitive Networking," in *Proc. of the 2017 VI Int. Conf. on Network, Communication and Computing*, ICNCC 2017, (New York, NY, USA), 2017.

[12] P. Heise, F. Geyer, and R. Obermaisser, "TSimNet: An Industrial Time Sensitive Networking Simulation Framework Based on OMNeT++," in *8th IFIP Int. Conf. on New Technologies, Mobility and Security (NTMS)*, 2016.

[13] Y. Chang and S. A. Ajila, "Design and implementation of experimental SPB network simulator on NS-3," in *26th IEEE Canadian Conf. on Electrical and Computer Engineering (CCECE)*, 2013.

[14] Mijeong Yang, Jinho Hahm, Youngsun Kim, and Sangha Kim, "Design and implementation of the IS-IS routing protocol with traffic engineering," in *9th Asia-Pacific Conf. on Communications*, 2003.

[15] D. Fedyk, D. Allan, P. Ashwood-Smith, N. Bragg, J. Farkas, M. Ouellete, M. Seaman, and P. Unbehagen, "RFC 6329: IS-IS Extensions Supporting IEEE 802.1aq SPB," tech. rep., Internet Engineering Task Force, 2012.

[16] D. Allan, P. Ashwood-Smith, N. Bragg, J. Farkas, D. Fedyk, M. Ouellete, M. Seaman, and P. Unbehagen, "Shortest Path Bridging: Efficient Control of Larger Ethernet Networks," *IEEE Communications Magazine*, no. 10, 2010.

Appendix I

On the Security of IEEE 802.1 Time-Sensitive Networking

Abstract

IEEE 802.1 Time-sensitive Networking (TSN) standards are envisioned to be the backbone of mission-critical networks in the near future. Such networks have strict latency and reliability requirements, and any missed deadline for critical services may have hazardous consequences. Low-latency and deterministic communication scheme in TSN makes the network potentially susceptive for various attacks. Therefore, protecting time-sensitive networks against security threats is a crucial design concern. In this paper, we discuss more than 30 potential security issues and threats of IEEE 802.1 TSN protocols. Our goal is to explore attack surfaces on such systems to be more extensively analyzed and eventually protected.

Reference

```
Doğanalp Ergenç, C. Brülhart, J. Neumann, L. Krüger, M. Fischer.
On the Security of IEEE 802.1 Time-Sensitive Networking.  IEEE In-
ternational Conference on Communications (ICC), Workshop on Time-
sensitive and Deterministic Networking, 2021.
```

Contribution

In the forementioned publication, the whole contribution belongs to this thesis. All co-authors helped to improve the quality of the paper with their valuable feedback.

On the Security of IEEE 802.1 Time-Sensitive Networking

Doğanalp Ergenç
Universität Hamburg
ergenc@informatik.uni-hamburg.de

Cornelia Brülhart
ZAL GmbH
cornelia.bruelhart@zal.aero

Jens Neumann
ZAL GmbH
jens.neumann@zal.aero

Leo Krüger
ZAL GmbH
leo.krueger@zal.aero

Mathias Fischer
Universität Hamburg, DE
mfischer@informatik.uni-hamburg.de

***Abstract*—IEEE 802.1 Time-sensitive Networking (TSN) standards are envisioned to be the backbone of mission-critical networks in the near future. Such networks have strict latency and reliability requirements, and any missed deadline for critical services may have hazardous consequences. Low-latency and deterministic communication scheme in TSN makes the network potentially susceptive for various attacks. Therefore, protecting time-sensitive networks against security threats is a crucial design concern. In this paper, we discuss more than 30 potential security issues and threats of IEEE 802.1 TSN protocols. Our goal is to explore attack surfaces on such systems to be more extensively analyzed and eventually protected.**

***Index Terms*—Time-Sensitive Networking, IEEE 802.1, Security**

I. Introduction

In several mission-critical domains such as aviation, automotive, and industrial networks, IEEE 802.1 Time-Sensitive Networking (TSN) standards are considered as an important building block to guarantee deterministic communication. The standards improve existing Ethernet protocols with real-time capabilities and offer a complete toolset for synchronization, management, and reliability of time-sensitive communication. These advantages, e.g., in the areas of quality of service (QoS), low latencies and bandwidth reservation may aid with implementing new superior technologies. Therefore, in the near future, TSN is promising to replace existing protocols in mission-critical domains, bringing a wider homogeneity and standardization in the networks.

In time-sensitive networks, *time* itself is an attack vector. A missed deadline in critical services due to an unexpected delay induced by an attacker can lead to severe consequences. Therefore, latency is not only the primary performance metric but a matter of safety and security. Various aspects of TSN such as time synchronization, scheduling of data traffic, and orchestration and reservation of network resources are appealing for the attackers to hinder deterministic communication where only a micro- to millisecond jitter is tolerable. As a result, the security of time-sensitive networks should be a prior design concern.

Although several studies review TSN [1], [2] and discuss the open issues of TSN protocols [3], [4] and security concerns of deterministic networking [5], there is not a comprehensive work addressing the security aspects of TSN protocols. Accordingly, in this paper, we discuss a multitude of possible threats against the TSN protocols. As a result, we list more than 30 threats for different classes of the protocols that constitute the main mechanisms such as scheduling, configuration, redundancy, and synchronization. In the rest of this paper, we present a summary of some TSN protocols in Section II. The threats against different mechanisms of TSN are discussed in Section III. Lastly, Section IV concludes the discussion and the potential extensions of this study.

II. IEEE 802.1 Time-Sensitive Networking

The collection of standards for TSN consists of base standards, such as the IEEE 802.1Q-2018, which contains sections to describe TSN transmission mechanisms for real-time applications, as well as ongoing standards for project improvement. The TSN standard set covers a wide range of quality of service (QoS) transmission requirements and can be classified in categories as seen in Fig. 1. Some of the main components in TSN networks are: End-points, bridges, as well as flows and are often used to describe communication mechanisms in various TSN protocols. Here, TSN bridges switch the Ethernet packets between end-points, which can either be a traffic generating instance, also called Talker, or the packet destination, also referred to as Listener. Finally, TSN flows describe the time-critical communication between end devices and are identified uniquely in the network. To give a short overview, the following section describes only some of the most prevalent industrially used TSN standards in their classification.

1) Time Synchronization: **IEEE 802.1AS Time Synchronization for Time Sensitive Applications (AS):** Network-wide deterministic communication and a uniform frame reference for all network participants require time synchronization of all TSN network entities. IEEE 802.1AS [6] gPTP ensures the desired time synchronization by applying the generic Precision Time Protocol (gPTP), which establishes a master-slave clock hierarchy between TSN network participants. The Best Master Clock Algorithm (BMCA) allows the selection of a network root timing reference, called grandmaster, or Clock Master (CM), which is defined as the bridge with the most

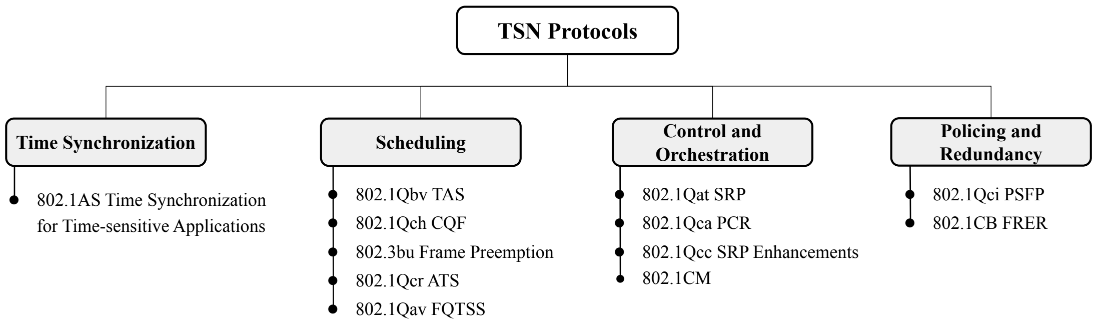


Fig. 1. IEEE 802.1 TSN protocols. It illustrates the published protocols excluding data models and the complementary protocols under IEEE 802.1

accurate clock. Following this, gPTP passes messages between CM and Clock Slaves (CSs) to calculate the propagation delay between CSs. The frame residence time of each CS also describes the required time for transmission from ingress to egress port. This allows for a time synchronization of the real-time clocks of gPTP devices, which are time-aware, as well as passive and active non-gPTP devices, which do not contribute to time synchronization and the BMCA of the TSN network.

2) Scheduling: **IEEE 802.1Qbv Enhancements to Traffic Scheduling: Time-Aware Shaper (TAS):** The time-aware Qbv [7] TAS scheduler partitions Ethernet network communication in time-windows of fixed length, to which one of eight Ethernet priorities can be assigned. Each priority receives its dedicated transmission window, which prevents traffic frame starvation and ensures the transmission medium reservation for high priority traffic. Additionally, to prevent the effects of blocking and overlapping of time-critical transmission windows, the TAS puts a time slice called guard band, before time-critical traffic. Applying preemptive frame transmission allows to reduce the guard band to the smallest Ethernet frame fragment and a more selective QoS. However, to guarantee that scheduled traffic can not be interfered by lower priority traffic, the Qbv requires all TSN network participants to be synchronized in time. The configuration of the TAS, such as the opening and closing of queue gates based on the current time can be defined by the Gate Control List (GCL), which is loaded statically before runtime.

IEEE 802.1Qch Cyclic Queuing and Forwarding (CQF): The Qch [8] allows a cyclic synchronized LAN bridge frame transmission without congestion loss and latency issues. Depending on the network topology and switches, worst-case delays may occur. These networks benefit therefore greatly from the synchronized enqueue and dequeue operations of the CQF, as it allows a synchronized transmission between bridges. Setting a worst-case deterministic delay of two times the cycle time between sender and receiver results in a topology independent latency.

IEEE 802.3br and 802.1Qbu Interspersing Express Traffic (IET) and Frame Preemption: The combination of these two standards [9] IET aims to decrease some of the drawbacks of the guard band, implemented by the TAS, which is, e.g., the potentially wasted transmission time. For this, the egress port is separated into two MAC service interfaces: Preemptable MAC (pMAC) and express MAC (eMAC). Express frames may preempt the preemptable frames, which are held on to the transmission medium to be transmitted after the express frame completion. This results in a reduced guard band down to the transmission time of the shortest low priority frame segment, as in the worst case the low priority frame fragment is completed before the start of the next high priority frame.

IEEE 802.1Qcr [10] and IEEE 802.1Qav [11] are the other TSN standards related to scheduling.

3) Control and Orchestration: **IEEE 802.1Qca Path Control and Reservation (PCR):** This standard [12] is based on Shortest Path Bridging (SPB) and specifies bridging on explicit paths. Multiple paths for unicast and multicast frame transmission can be determined, depending on the topologies, e.g. internal spanning tree (IST), equal cost tree (ECT) or shortest path. After collecting any network topology information from nodes to find redundant paths, the PCR instruments the necessary bandwidth and resource reservation for redundant and optimal network traffic transmission.

IEEE 802.1Qat Stream Reservation Protocol (SRP) and IEEE 802.1Qcc Enhancements to SRP and Centralization Management: Guaranteed network resources along transmission paths and acceptance or rejection of flows are provided through the admission controls of the IEEE 802.1Qat SRP. This distributed P2P protocol allows the reservation of network resources and stream advertisement as required by the QoS requirements in packet switched networks. The SRP utilizes the Multiple Registration Protocol (MRP), which identifies and registers traffic streams by a 64-bit unique identifier, called StreamID. It consists of the 48-bit Extended Unique Identifier (EUI-48) concatenated with a 16-bit UniqueID. Resources for a stream are reserved by the SRP based on the latency traffic class and bandwidth requirements through the use of three signaling protocols: Multiple MAC Registration Protocol (MMRP), Multiple VLAN Registration Protocol (MVRP), and Multiple Stream Registration Procol (MSRP). These protocols supervise the VLAN membership and group registration propagation, as well as distributed network resource reservation and data stream advertisement. IEEE 802.1Qcc enhances SRP with additional network tools for a global network management. It adds a User Network Interface (UNI) for the requesting of link layer services and a Centralized Network Configuration (CNC) node for a centralized resource reservation and scheduling to the SRP. The CNC interacts with the UNI by using remote management protocols, such as NETCONF/RESTCONF and offers data modeling compatability to, e.g. IETF YANG/NETCONF. IEEE 802.1Qcc allows three different configuration and resource management models. In the fully centralized model, a talker provides stream requirements to the CNC that calculates and selects possible transmission time-slots. The decentralized model relies on the message exchange between nodes and not on a centralized control for the path setup. Lastly, the hybrid model utilizes SRP by the end-points for requirement advertisement but where the CNC may still be used for reservation coordination.

4) Policing and Redundancy: **IEEE 802.1Qci Per-Stream Filtering and Policing (PSFP):** Only a few TSN standards aim at increasing security within the network. Among those, IEEE 802.1Qci PSFP [13] improves robustness by policing ingress data flows. Here, rule matching and per-flow filtering prevents traffic overload and Denial of Service (DoS) attacks. PSFP identifies streams by individual StreamIDs and improves network robustness by matching frames on ingress ports with specified stream IDs and priority levels. All streams are coordinated by the PSFP stream gate, which allows the subsequent application of other policy actions such as dropping or queuing a packet.

IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER): Additional improvements of reliable TSN frame transmission and network reliability can be enforced by employing the IEEE 802.1CB FRER [14] standalone standard. FRER allows frame redundancy of critical traffic by sending duplicate frames over disjoint paths. It also provides mechanisms to eliminate packet duplicates, if both frames reach their destination. This prevents congestions loss, as well as package loss, e.g., due to path failure. Packet replication can be further configured, by assigning traffic classes, while path information is selected through the TSN stream identification and a sequence generation function, to save frame redundancy information in the Redundancy Tag (R-TAG). FRER packets are handy for critical traffic transmission, as frame sequence numbers and timing information is also needed to limit the required memory for duplicate frame elimination.

III. TSN Threats

We have reviewed several TSN threats from the academic literature and the industrial talks and extended those with the threats we envision following the well-known threat modeling framework, STRIDE [15]. STRIDE covers the threats **S**poofing (red), **T**ampering (orange), **R**epudiation, **I**nformation disclosure, **D**enial of service (blue), and **E**levation of privilege (green) threats against system components and is used as a guideline for secure system design. Here, we do not consider repudiation and information disclosure threats. Nevertheless, some of the given threats can fall into multiple categories, including repudiation and information disclosure threats. We have then categorized them according to the classification of TSN protocols in Section II to give a complete view of the TSN mechanisms from a security perspective. Fig. 2 summarizes the review of the threats.

Note that we do not assume a particular attacker model for our threat model. Therefore, we review the threats within a broader scope from more destructive DoS attacks to subtle attacks that can result in QoS degradation. In the rest of the section, those threats are explained in detail.

A. Time Synchronization Threats

Time synchronization protocols have been widely used in various systems and domains and studied in depth [16]–[18], including their security aspects [4], [19], [20]. There are also further attempts specifying the requirements and attack scenarios on such mechanisms [5], [21].

Even though IEEE 802.1AS simplifies the configuration of the standard PTP, it still inherits several attack vectors. Since it requires a similar deployment with PTP, there should be a grandmaster(s) taking the role of a master clock that distributes the reference time to the slave nodes, i.e., other nodes syncing to the common time. Here, **a compromised or impersonated grandmaster** results in spreading false or inconsistent timing information. The impersonation, for instance, might be conducted **intervening the distributed grandmaster election process** while nodes announce their priority to be selected as the master clock. Similarly, **unauthorized joins as a master clock** bypassing the election may override the elected grandmaster. In the case of a usage of a single clock master, **sabotaging the master clock** may result in a short asynchronization until a new master is elected or switching over a redundant (passive). For such cases, multiple redundant master clocks can be selected.

Targeting the master clock is challenging as it is usually one of the protected assets in the network. **Tampering and forging time synchronization packets** is another way to intervene in the time synchronization process. **Altering the timestamps** is the most straightforward attack after tampering [19], which can be partially avoided by encryption and integrity protection of the packets. However, meta-information such as protocol specifications can still be neglected as confidential information. For instance, as PTPv2 is not backward compatible with PTPv1, **mixing protocol version specifications** can hinder a consistent time synchronization process in the overall network. Moreover, even though cryptography is a solution to protect against malicious packet modifications, it is not effective against **delaying attacks** that result in incorrect measurements of hop-to-hop latency, which is used to readjust timing information eliminating propagation delays [22].

B. Scheduling Threats

IEEE 802.1Qbv TAS is the main protocol to establish deterministic communication via scheduling. It requires the configuration of port-based gate control lists that can be

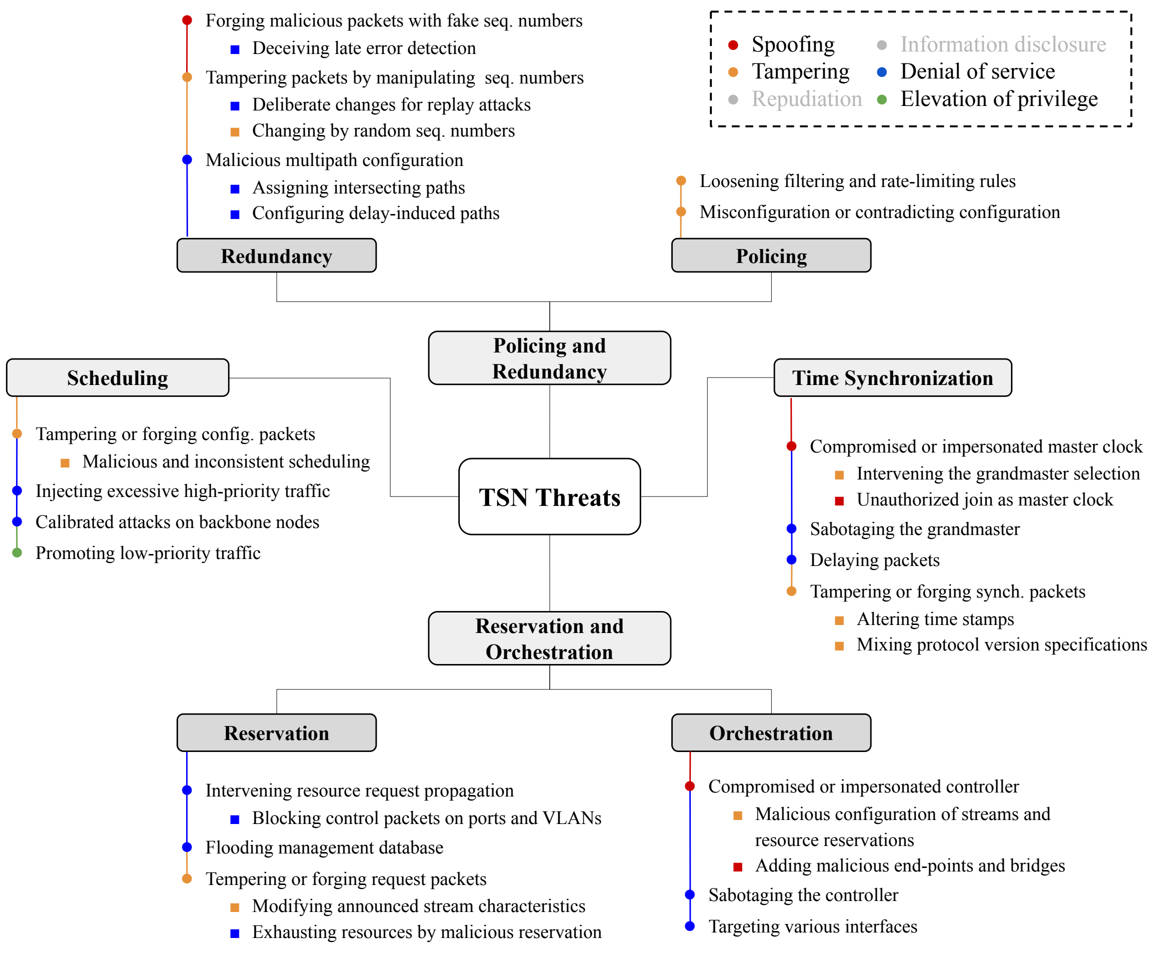

Forging malicious packets with fake seq. numbers
Deceiving late error detection
Tampering packets by manipulating seq. numbers
Deliberate changes for replay attacks
Changing by random seq. numbers
Malicious multipath configuration
Assigning intersecting paths
Configuring delay-induced paths
Redundancy
Spoofing
Tampering
Repudiation
Information disclosure
Denial of service
Elevation of privilege
Loosening filtering and rate-limiting rules
Misconfiguration or contradicting configuration
Policing
Policing and Redundancy
Scheduling
Tampering or forging config. packets
Malicious and inconsistent scheduling
Injecting excessive high-priority traffic
Calibrated attacks on backbone nodes
Promoting low-priority traffic
TSN Threats
Time Synchronization
Compromised or impersonated master clock
Intervening the grandmaster selection
Unauthorized join as master clock
Sabotaging the grandmaster
Delaying packets
Tampering or forging synch. packets
Altering time stamps
Mixing protocol version specifications
Reservation and Orchestration
Reservation
Intervening resource request propagation
Blocking control packets on ports and VLANs
Flooding management database
Tempering or forging request packets
Modifying announced stream characteristics
Exhausting resources by malicious reservation
Orchestration
Compromised or impersonated controller
Malicious configuration of streams and resource reservations
Adding malicious end-points and bridges
Sabotaging the controller
Targeting various interfaces

Fig. 2. TSN threats matching with corresponding STRIDE threats

configured locally or via an external controller, whose details are given in IEEE 802.1Qcc. Such a remote configuration can be a subject to **packet tampering and forgery on the management protocol** such as Network Configuration Protocol (NETCONF) or Simple Network Management Protocol (SNMP). Even though those protocols can operate on the top of the secure protocols like Secure Shell (SSH) and Transport Layer Security (TLS) [23], [24], older password-based authentication schemes are still valid and pose risks. As a result, while an attacker can assign higher-priority and thus more resources to low-priority traffic, it can also reduce the bandwidth for critical streams, which quickly leads to missed deadlines due to congestion.

Assuming that the attacker has access to a TSN bridge, there is no mechanism integrated into the scheduling protocols to validate if it satisfies the QoS requirements and even ensures that no service class is blocked, i.e., never having open gates. Therefore, a lack of validation mechanism can lead to **malicious and inconsistent scheduling** that is hard to detect. Even though the scheduling validation tools have become popular, especially in the automotive industry [25], [26], they are only performed in the initial network design stage.

It is also possible to abuse the existing gate list configuration without accessing the actual configuration. For instance, **injecting excessive high-priority traffic** creates high congestion and queueing delays that can lead to packet drops and missed deadlines. Whereas injection attacks can threaten scheduling mechanisms, the bucketing logic of some of the scheduling methods can be utilized as a countermeasure to limit the impact of congestion due to those attacks [27]. However, especially in preemptive mode [9], this congestion may also interrupt low priority traffic. Similarly, **promotion of low-priority frames to high-priority ones** (assuming that configured VLAN and priority matchings are known by the attacker) can hinder given scheduling scheme.

Lastly, as TSN scheduling provides at most determinism for end-to-end communication, it is easier to conduct **calibrated attacks on tightly-scheduled nodes** for an attacker [28]. That is, the frames of high-priority streams are known to (i) when being forwarded by a particular TSN node or (ii) where being processed at the given time since there is less room for unexpected delays, out of order packets, rerouting, etc. in TSN. Even though it is not a threat to the scheduling mechanisms per se, it is now a threat caused by tight scheduling and deterministic communication. Therefore, the scheduling configuration is a valuable system design parameter.

C. Control and Orchestration Threats

Attacks towards TSN orchestration and reservation mechanisms can hinder the allocation of required resources in terms of bandwidth and time-scheduling, as well as redundant resources for reliability. IEEE 802.1Qat SRP is a relatively mature TSN protocol that has been revised since the design of Audio Video Bridging (AVB) systems. In the standard, a number of security considerations are addressed [29]. As SRP depends on the propagation of talker stream advertisements, where the required resources and the stream characteristics are announced to be accepted by the listener and also to arrange network resources, any compromised bridge **intervening the advertisement propagation** can block forwarding of the frames for the respective stream. Similarly, a malicious configuration of a bridge to **block control packets on particular ports or forwarding to certain VLANs** can disturb the communication for only targeted domains of a network. Instead of an intervention, tampering the announcement packets to **modify maximum frame size and packet sending frequency** of a stream can mislead other bridges to miscalculate the expected latency, which eventually causes missed deadlines.

Apart from manipulating the existing stream announcements, an attacker can **forge stream reservation requests** as a malicious talker to exhaust available resources. Such malicious requests can threaten the internal resources of bridges by **flooding limited size of management information database** (MIB) where the stream registration objects are stored. Flooding attacks are common threats for legacy protocols like the Address Resolution Protocol (ARP) and can be effective in case of lack of rate-limiting and filtering on bridges.

TSN offers more than the distributed stream reservation protocol and enables network-wide orchestration and management with centralized controlling schemes and end-to-end path reservations. IEEE 802.1Qcc [30] proposes two other centralized controlling schemes other than the distributed one, which directly leverages SRP. The new centralized schemes inherit the characteristics of a software-defined networking (SDN) architecture [31] including its security considerations [32], [33]. The controller is responsible for the configuration of almost all TSN protocols, except the time-synchronization mechanism [30]. Therefore, a **a compromised or impersonated controller** can have a disastrous impact in the overall TSN system. While it can result in **malicious misconfiguration of streams, reserved resources, or end-points**, an attacker can conduct more hazardous attacks by completely **sabotaging the controller** for a DoS. For such scenarios, even though TSN standards do not specify the design of multiple decentralized controllers, redundancy of the controller can help against a single point of failure. Note that even subtle changes in the configuration, for example, re-routing critical streams to more delay-prone paths, can affect the system in long-term by degrading the QoS or causing a missing deadline. Moreover, indirect threats, i.e., not against the orchestration but using orchestration mechanisms, such as **adding new malicious end-points or bridges** by using controller can enable multi-step propagating attacks [5].

A centralized controller requires different interfaces to enable configuration between (i) the end-points and the centralized configuration unit (CUC), which has an important role in centralizing SRP protocol in one of the controlling schemes, (ii) bridges and the centralized network controller (CNC), and (iii) CUC and CNC. Consequently, there is an increased attack surface where an attacker can **target multiple interfaces between end-points, bridges, and controller units**. Note that while (i) inherits the threats against SRP, (ii) and (iii) correspond to the southbound and northbound interfaces of SDN, respectively and thus may suffer from similar threats.

D. Policing and Redundancy Threats

IEEE 802.1CB FRER does not have any built-in security mechanism and thus its two main functions, replication and elimination, are open to particular threats. Some studies in the literature have addressed possible problems of FRER [3], [34]. Several vulnerabilities have also been identified in the correspondent redundancy protocols operating above link layer [5] and having similar principles with FRER.

The elimination mechanism of FRER allows a bridge to forward the first copy of the processed packet and drops the second copy, i.e., replicated for redundancy, when it is received by the same bridge. An attacker can **forge malicious packets with fake sequence numbers** so that the original packets with the same sequence numbers are dropped. Similarly, even if an attacker sends forged packets later than original packets, it can **deceive the late error detection mechanism** by introducing forged replicas to compensate, e.g., a failed or compromised redundant stream. That is, even though the redundant communication is stonewalled by the attacker, it can still create an illusion of a working redundancy mechanism with forged packets containing no actual data. Apart from such calibrated attacks, **tampering packets by changing their sequence numbers randomly** can easily induce unexpected packets drops and delivery of out-of-order packets to the listener. **Changing the sequence number of a replica packet to forward it again** can lead to replay attacks as well.

The FRER standard defines a set of functions to be implemented and configured by network designers. Its proper configuration and integration to the packet processing pipeline of TSN bridges are on-going issues. For instance, FRER does not guarantee in-order delivery of multiple member streams. As a result, an attacker can **maliciously configure multiple paths of redundant streams to induce extra delay to the redundant paths and trigger out-of-order packet delivery** that hinder seamless redundancy in case of failures and cause degradation in QoS. Another issue of path selection is related to the elimination mechanism. While other redundancy mechanisms of Ethernet guarantee the existence of disjoint paths, e.g., through isolated networks or a ring topology [35], FRER eliminates duplicates to ensure that no copies are forwarded by the same bridge. As a result, **rerouting the stream through intersecting paths** can result in the unexpected elimination of packets and decrease intended level of redundancy [3].

Lastly, IEEE 802.1Qci PSFP offers data plane filtering and policing mechanism. Similar to FRER, it coexists with other reliability protocols and filtering mechanisms of bridges. Therefore, its **misconfiguration configuration with the other protocols** can lead some packets to be unintentionally filtered out or the malicious ones to be forwarded. Similarly, by **loosening filtering and rate limiting rules**, an attacker can launch more impactful DoS attacks, e.g., by flooding, in the next step. However, at the end, PSFP is designed to protect the network and TSN devices, while its configuration is the only protection mechanism among the TSN standards.

IV. Conclusion

After the explanation of some of the most prominent TSN standards, we present more than 30 threats to state possible attacks for the mentioned standards and consequently offer insight about potential weaknesses in TSN networks. For future work, we would like to discuss potential countermeasures and attack mitigation techniques in concrete scenarios and develop strategies to strengthen TSN networks against various attacker types. Additionally, we plan to develop different attacker models, e.g., the different capabilities of an internal or external attacker, to evaluate the described threats in more detail.

References

[1] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, 2019.

[2] L. Lo Bello and W. Steiner, "A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems," *Proc. of the IEEE*, vol. 107, no. 6, 2019.

[3] D. Ergenc and M. Fischer, "On the Reliability of IEEE 802.1CB FRER," in *IEEE Int. Conf. on Computer Communications (INFOCOM)*, 2021.

[4] C. DeCusatis, R. M. Lynch, W. Kluge, J. Houston, P. A. Wojciak, and S. Guendert, "Impact of Cyberattacks on Precision Time Protocol," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 5, 2020.

[5] T. Mizrahi and E. Grossman, "Deterministic Networking (DetNet) Security Considerations," Internet-Draft draft-ietf-detnet-security-11, Internet Engineering Task Force, 2020. Work in Progress.

[6] "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications," *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pp. 1–421, 2020.

[7] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: IEEE 802.1Qbv-2015 Enhancements for Scheduled Traffic," 2015.

[8] "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 29: Cyclic Queuing and Forwarding," pp. 1–30, 2017.

[9] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption," 2016.

[10] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks - Amendment 34: IEEE 802.1Qcr-2020 Asynchronous Traffic Shaping," 2020.

[11] "IEEE Standard for Local and metropolitan area networks– Virtual Bridged Local Area Networks Amendment 12: IEEE 802.1Qav-2009 Forwarding and Queuing Enhancements for Time-Sensitive Streams," 2010.

[12] "IEEE Standard for Local and Metropolitan Area Networks Bridges and Bridged Networks Amendment 24: IEEE Std 802.1Qca-2015 Path Control and Reservation," March 2016.

[13] "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: IEEE 802.1Qci-2017 Per-Stream Filtering and Policing," 2017.

[14] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks– IEEE Std 802.1CB-2017 Frame Replication and Elimination for Reliability," Oct 2017.

[15] A. Shostack, S. Hernan, S. Lambert, and T. Ostwald, "Uncover Security Design Flaws Using The STRIDE Approach."

[16] W. Jingchao, Z. Ruohan, and G. Weiwen, "Time Synchronization in Networks: A Survey," in *Proc. of the 2nd Int. Conf. on Control and Computer Vision*, ICCCV 2019, (New York, NY, USA), Association for Computing Machinery, 2019.

[17] D. Fontanelli and D. Macii, "Accurate time synchronization in PTP-based industrial networks with long linear paths," in *IEEE Int. Symp. on Precision Clock Synchronization for Measurement, Control and Communication*, 2010.

[18] H. Lim, D. Herrscher, L. Völker, and M. J. Waltl, "IEEE 802.1AS time synchronization in a switched Ethernet based in-car network," in *IEEE Vehicular Networking Conf. (VNC)*, 2011.

[19] B. Moussa, C. Robillard, A. Zugenmaier, M. Kassouf, M. Debbabi, and C. Assi, "Securing the Precision Time Protocol (PTP) Against Fake Timestamps," *IEEE Communications Letters*, vol. 23, no. 2, 2019.

[20] E. Shereen, F. Bitard, G. Dán, T. Sel, and S. Fries, "Next Steps in Security for Time Synchronization: Experiences from implementing IEEE 1588 v2.1," in *IEEE Int. Symp. on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2019.

[21] T. Mizrahi, "Security requirements of time protocols in packet switched networks," RFC 7384, October 2014.

[22] M. Ullmann and M. Vögeler, "Delay attacks — Implication on NTP and PTP time synchronization," in *2009 Int. Symp. on Precision Clock Synchronization for Measurement, Control and Communication*, 2009.

[23] M. Wasserman, "Using the NETCONF Protocol over Secure Shell (SSH)," RFC 6242, June 2011.

[24] M. Badra, A. Luchuk, and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication," RFC 7589, June 2015.

[25] O. Creighton, N. Navet, P. Keller, and J. Migge, "Towards Computer-Aided, Iterative TSN-and-Ethernet-based E/E Architecture Design," in *IEEE-SA Ethernet & IP at Automotive Technology Day*, 2020.

[26] R. Oliveira, I. Raghupatruni, D. Martini, and A. Henkel, "A Framework to Virtually Validate QoS Contracts in Ethernet-based Vehicle Data Infrastructures for Automotive Cyber-physical Systems," in *IEEE-SA Ethernet & IP at Automotive Technology Day*, 2020.

[27] P. Meyer, T. Häckel, F. Korf, and T. C. Schmidt, "DoS Protection through Credit Based Metering–Simulation Based Evaluation for Time-Sensitive Networking in Cars," *arXiv preprint arXiv:1908.09646*, 2019.

[28] Hummen, René and Kleineberg, Oliver, "Cyber-security for Modern TSN Automation Networks," *Industrial Ethernet Book*, vol. 104, 2018.

[29] "IEEE Standard for Local and metropolitan area networks–Virtual Bridged Local Area Networks Amendment 14: IEEE Std 802.1Qat-2010 Stream Reservation Protocol (SRP)," 2010.

[30] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: IEEE Std 802.1Qcc-2018 Stream Reservation Protocol (SRP) Enhancements and Performance Improvements," 2018.

[31] S. B. H. Said, Q. H. Truong, and M. Boc, "SDN-Based Configuration Solution for IEEE 802.1 Time Sensitive Networking (TSN)," *SIGBED Rev.*, vol. 16, Feb. 2019.

[32] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in Software Defined Networks: A Survey," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, 2015.

[33] A. Demirpolat, D. Ergenç, E. Ozturk, Y. Ayar, and E. Onur, "Software-defined network security," in *Enabling Technologies and Architectures for Next-Generation Networking Capabilities*, IGI Global, 2019.

[34] R. Hofmann, B. Nikolić, and R. Ernst, "Challenges and Limitations of IEEE 802.1CB-2017," *IEEE Embedded Systems Letters*, vol. 12, no. 4, 2020.

[35] "Industrial communication networks - High availability automation networks - Part 3: IEC 62439-3:2016 RLV Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR)," March 2016.

Appendix J

TSNZeek: An Open-source Intrusion Detection System for IEEE 802.1 Time-sensitive Networking

Abstract

IEEE 802.1 Time-sensitive Networking (TSN) standards are envisioned to replace legacy network protocols in critical domains to ensure reliable and deterministic communication over off-the-shelf Ethernet equipment. However, they lack security countermeasures and can even impose new attack vectors that may lead to hazardous consequences. This paper presents the first open-source security monitoring and intrusion detection mechanism, TSNZeek, for IEEE 802.1 TSN protocols. We extend an existing monitoring tool, Zeek, with a new packet parsing grammar to process TSN data traffic and a rule-based attack detection engine for TSN-specific threats. We also discuss various security-related configuration and design aspects for IEEE 802.1 TSN monitoring. Our experiments show that TSNZeek causes only ∼5% CPU overhead on top of Zeek and successfully detects various threats in a real TSN testbed.

Reference

`Doğanalp Ergenç, R. Schenderlein, M. Fischer. TSNZeek: An Open-source Intrusion Detection System for IEEE 802.1 Time-sensitive Networking. IFIP Networking Conference, International Workshop on Time-Sensitive and Deterministic Networking (TENSOR), 2023.`

Contribution

In the forementioned publication, the main contributions of this thesis are designing the overall IDS module, specifying attacks and their respective notices, and conducting the evaluation. The second co-author implemented several parts of TSNZeek in the context of a master's thesis. The third co-author helped to improve the quality of the paper with his valuable feedback.

TSNZeek: An Open-source Intrusion Detection System for IEEE 802.1 Time-sensitive Networking

Doğanalp Ergenç, Robin Schenderlein, Mathias Fischer
University of Hamburg, Germany
name.surname@uni-hamburg.de

***Abstract*—IEEE 802.1 Time-sensitive Networking (TSN) standards are envisioned to replace legacy network protocols in critical domains to ensure reliable and deterministic communication over off-the-shelf Ethernet equipment. However, they lack security countermeasures and can even impose new attack vectors that may lead to hazardous consequences. This paper presents the first open-source security monitoring and intrusion detection mechanism, TSNZeek, for IEEE 802.1 TSN protocols. We extend an existing monitoring tool, Zeek, with a new packet parsing grammar to process TSN data traffic and a rule-based attack detection engine for TSN-specific threats. We also discuss various security-related configuration and design aspects for IEEE 802.1 TSN monitoring. Our experiments show that TSNZeek causes only $\sim$5% CPU overhead on top of Zeek and successfully detects various threats in a real TSN testbed.**

***Index Terms*—intrusion detection, security, IEEE 802.1 TSN**

I. Introduction

Modern mission-critical systems are composed of several interconnected components and services that require reliable and time-sensitive communication. To satisfy such requirements and reduce the dependency on domain-specific networking technologies, IEEE 802.1 Time-sensitive Networking (TSN) task group has proposed a set of standards. These standards enable low-latency, fault-tolerant, and deterministic communication on top of standard Ethernet protocols. However, IEEE 802.1 TSN protocols also induce several security threats across the domains and systems deploying them [1]–[3]. Timely detection of such threats is crucial, especially in safety-critical systems, in which a successful attack may lead to hazardous results.

Since the TSN standards are relatively new and prioritize the tight quality of service (QoS) requirements of critical systems, there is no comprehensive security solution against TSN-specific threats. Although a built-in traffic policing protocol is a part of the standards [4], it provides limited filtering capabilities to ensure that critical data streams receive sufficient resources. However, an effective solution requires monitoring time-sensitive streams over TSN protocols and recognizing malicious attempts. Accordingly, in this paper, we introduce an open-source network monitoring and intrusion detection system for IEEE 802.1 TSN protocols, `TSNZeek`[1]. We extended an existing monitoring tool, Zeek (former Bro [5]), to analyze the new TSN protocols and detect several TSN-specific attacks. Zeek is a well-established open-source network monitoring solution that is popularly deployed in real systems as well as used in academia for research purposes[2]. We focus on the threats against two TSN protocols, IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER) and IEEE 802.1Qcc Stream Reservation Protocol (SRP), since they (i) are the critical protocols addressing communication reliability and configuration, and (ii) have specific network behavior with new packets types and architectural aspects. Our contributions are listed as follows:

- We implement a new packet parser using a grammar definition language, *spicy*, to process SRP and FRER traffic via Zeek. To the best of our knowledge, this renders `TSNZeek` the first TSN-aware security monitoring tool.
- We implement an intrusion detection engine connected to Zeek to recognize several attacks described in our previous paper [1].
- We test our proposal in a TSN testbed and confirm that it successfully detects various threats with negligible overhead.
- We publish `TSNZeek` open-source together with the traffic and attack generation tools[3].

The remainder of this paper is organized as follows. Section II introduces the TSN protocols that `TSNZeek` is capable of processing, SRP and FRER. Section III presents the related work. Section IV describes the design and implementation of `TSNZeek`. Section V gives the experiment setup and evaluation. Lastly, Section VI concludes the paper.

II. Background on IEEE 802.1 TSN

This section describes two TSN protocols: IEEE 802.1Qcc SRP and IEEE 802.1CB FRER. In comparison to other TSN protocols, SRP and FRER introduce their own interfaces, packet structures, and configuration schemes that impose several security threats. Therefore, we mainly focus on them in this study.

A. IEEE 802.1Qcc Stream Reservation Protocol (SRP)

IEEE 802.1Qcc SRP introduces the resource reservation routines for time-sensitive streams to configure all TSN components in the systems satisfying tight QoS requirements. It proposes two main components: (i) a network configuration (CNC) entity to configure the TSN bridges remotely and (ii) a user configuration (CUC) entity to discover the

[1]A preprint of this paper is available at arXiv with the same title.

[2]Zeek Project, https://zeek.org

[3]The source code is available at https://github.com/UHH-ISS/tsnzeek

endpoints [6]. It further offers three configuration schemes utilizing those entities.

- In the **fully centralized model**, endpoints directly communicate with CUC over a user/network interface (UNI) and request network resources for TSN streams with certain requirements such as the worst-case latency and inter-arrival times. CNC then configures the bridges according to the requests received by CUC.
- In the **centralized network/distributed user model**, edge TSN bridges, e.g., bridges that endpoints are directly attached to, forward SRP requests to CNC with network-wide visibility. Similar to the fully centralized model, it is responsible for configuring all TSN bridges.
- In the **distributed model**, TSN bridges forward SRP requests to each other to handle configurations individually.

SRP imposes a complex packet structure that allows an endpoint to specify various requirements via type-length-value (TLV) fields and recursive header groups. The respective standard [6] at Section 35.2 (p.105-134) explains the whole packet structure in detail.

B. IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER)

IEEE 802.1CB FRER enables redundancy against link failures by sending duplicate TSN flows, which are called member streams [7]. The talker sends member streams through multiple redundant paths configured in advance. An incremental sequence number is embedded in FRER frames within the R-TAG header, and the duplicate frames across the member streams have the same sequence number. The member streams rejoin at one or more points (e.g., at the listener or an edge bridge) in the network, where duplicate frames are discarded by their sequence number. Finally, the listener receives the original compound stream.

To discard the duplicate frames and obtain the original stream, FRER utilizes various stream recovery functions. These functions consider the sequence number of the most recently received frame to perform frame elimination. For instance, the match recovery function eliminates all the frames with a sequence number smaller than the recently observed one. The frame elimination helps to drop the duplicate packets received due to stuck senders or misrouting.

III. Related work

In this section, we briefly present related work that investigates the security threats against IEEE 802.1 TSN protocols and proposes security solutions.

Regarding security threats, in [8], the authors discuss the security threats in TSN-based industrial control systems. In [9], they analyze the impact of denial-of-service attacks on TSN protocols. In [1], we listed more than 30 attack vectors against several TSN mechanisms.

Some of the existing TSN protocols can be utilized against such security threats. For example, in [3], the authors employ IEEE 802.1Qav Credit-based Shaper (CBS) to prevent denial of service attacks. The authors of [10] and [2] combine IEEE 802.1Qci Per-Stream Filtering and Policing (PSFP) protocol [4] with a centralized controller to enforce ingress policies for packet inter-arrival times and rates, and stream bandwidth. Similarly, in [11], the effectiveness of PSFP is analyzed for the security of TSN-based automotive networks. Lastly, in [12], the authors discuss security policies via PSFP enforced by a centralized policy server.

There are more practical design and implementation efforts for the monitoring and protection of time-sensitive systems. In [13], the authors propose a monitoring system for the bridge and link status as well as time-synchronization accuracy, excluding security and intrusion detection aspects. [14] presents a security module to improve TSN protocols with hardware encryption and authentication. IEEE 802.1AE Media Access Control (MAC) standard also enables authentication, integrity, and confidentiality in Ethernet-based data traffic [15].

None of the works above offers security monitoring or an IDS for IEEE 802.1 TSN protocols with specific packet structures, traffic characteristics and requirements, and thus security threats. In contrast, we propose an open-source and extendable IDS solution to address TSN-specific attacks.

IV. TSNZeek: Design and Implementation

`TSNZeek` consists of monitoring and intrusion detection components shown in Fig. 1. The monitoring component processes and log the received TSN traffic. The intrusion detection component obtains the processed frames from the monitoring component and implements the attack recognition logic for TSN-specific attacks.

The overall operation of `TSNZeek` can be described as follows: the *event engine* distinguishes the incoming packets by their EtherType values, which is a standard header type of Ethernet frames. Then, *TSN parser* processes SRP and FRER packets according to the new parsing grammar that we developed. After parsing, the *broker* disseminates those frames to the *notice engine* and *detection engine*. While the former engine logs the traffic and specific events using built-in Zeek functionalities, and the latter implements our attack detection rules. The detection engine then pushes notices back to the notice engine when an attack is recognized.

In the rest of this section, we elaborate on those modules together with the notices and alerts that `TSNZeek` can raise.

A. Monitoring Component

The monitoring component corresponds to the original Zeek in terms of its working dynamics. It consists of the event engine, the TSN parser, the broker, and the notice engine. The TSN parser (blue, dashed lines in Fig.1) has been implemented from scratch. The event and notice engines (red, dashed, and dotted lines) are existing Zeek components that we extended further. For the engines, we used Zeek v4.2.0.

1) Event Engine: The event engine registers the protocol analyzers during initialization to specify the parsing grammar for respective protocols. Fig. 2 shows a sample registration process for SRP and FRER. When an SRP or FRER frame is recognized by its Ethernet header (e.g., with type `0x22EA` and

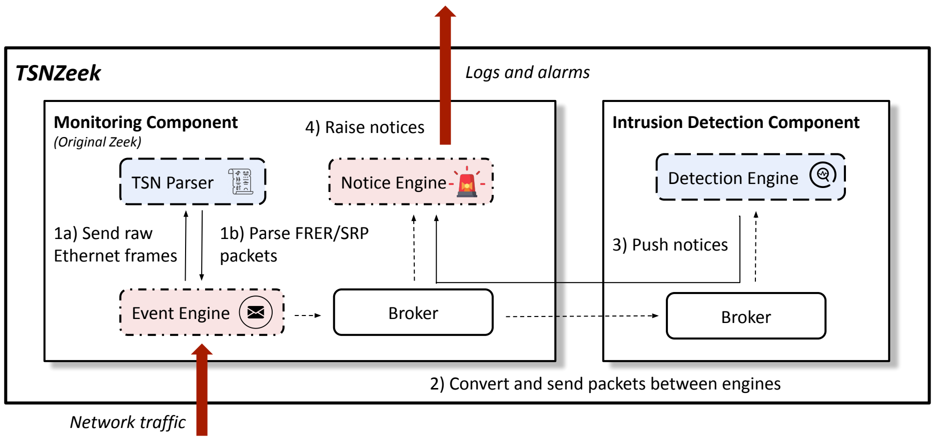


Fig. 1: The overview of TSNZeek. The blue/dashed blocks have been implemented from scratch. The red/dotted blocks are existing Zeek modules that we have extended and reconfigured.

0xF1C1, respectively), the event engine calls the respective packet parsing function implemented in the *TSN parser*.

```
event zeek_init() {
    if (!PacketAnalyzer::try_register_packet_analyzer_by_name("Ethernet",
            0xF1C1, "spicy::FRER"))
        print "Cannot register IEEE 802.1CB FRER analyzer";
    if (!PacketAnalyzer::try_register_packet_analyzer_by_name("Ethernet",
            0x22EA, "spicy::SRP"))
        print "Cannot register IEEE 802.1Qcc SRP analyzer"; }
```

Fig. 2: Registration of packet parsers.

The event engine registers further events to trigger logging facilities and inter-module packet dissemination via the broker. Both use the frame content provided by the protocol analyzers, i.e., for the received and parsed SRP and FRER frames. In the source code, all event registrations are implemented in the scripts *main.zeek* and *tsn.zeek* via Zeek scripting language.

2) TSN Parser: This module introduces the parsing functions for complex packet structures with many recursive header types. We implemented the parser using *spicy* v1.4.0, which is a grammar generation framework for network protocols and file formats[4]. We follow the packet definitions in the standards of SRP and FRER so that our parser can process any TSN traffic originated from a standard-compliant talker.

Fig. 3 shows an example *spicy* function to parse the talker information from an SRP frame. It extracts various traffic specifications and requirements given by a talker. Moreover, we define several FRER and SRP-specific header types, e.g., FRER R-TAG for sequence numbers, to be used within the parsing functions. In the source code, the files with *spicy* extension define the header types and parsing functions.

3) Broker: The broker is the built-in publish/subscribe messaging framework of Zeek. We implement three event topics in the broker: FRER, SRP talker, and SRP listener. Those topics are defined in *TSN.evt* in the source code. Whenever a respective type of frame is received, the broker publishes

[4]Zeek spicy, https://docs.zeek.org/projects/spicy

```
function makeTalker(obj: SRP::Talker): TSNZeek::Talker {
    local lendStationInterfaces = makeEndStationInterfaces(obj.
        endStationInterfaces);
    local ldataFrameSpecification = makeDataFrameSpecification(obj.
        dataFrameSpecification);
    local ltSpecTimeAware = makeTSpecTimeAware(obj.tSpecTimeAware);
    local luserToNetworkRequirements =
        makeUserToNetworkRequirements(obj.
        userToNetworkRequirements);
    local linterfaceCapabilities = makeInterfaceCapabilities(obj.
        interfaceCapabilities);
    return (
      makeStreamID(obj.streamID), makeStreamRank(obj.streamRank),
      lendStationInterfaces, ldataFrameSpecification,
      makeTrafficSpecification(obj.trafficSpecification),
      ltSpecTimeAware, luserToNetworkRequirements,
      linterfaceCapabilities); }
```

Fig. 3: Parsing function for SRP talker group.

its content, which is provided by the parser. The notice and detection engines subscribe to those topics and obtain the content of the frames for further analysis accordingly. For TSNZeek, we used Zeek Broker v2.2.0.

4) Notice Engine: The notice engine flags certain security events and logs received TSN traffic. We used the built-in notice facility of Zeek for this module. We configure the notice engine to log the received FRER and SRP frames partially to avoid an excessive amount of logs. A security event could be an anomaly in the configuration and network behavior, or a detected attack. The detection engine recognizes those events and then respective notice alerts are raised by the notice engine. The available notices are listed as follows.

- **N1.SRP. Excessive resource request:** If any talker demands more network resources than a predefined threshold, the notice engine raises this notice.
- **N2.SRP. Deviating resource request:** This notice alerts the resource demands that are marginally different from the previous SRP requests as it may indicate a malicious reservation.
- **N3.SRP. Too many requests:** It alerts if too many requests are received in a time interval, as it can indicate a stuck talker or an attack to exhaust the network resources.
- **N4.SRP. Changing existing allocation:** This notice alerts

in case of an attempt to change an existing SRP reservation.

- **N5.SRP. Dangling resources:** This notice alerts the dangling resources if they are still not used after a predefined time after their registration, as it may indicate a faulty endpoint.
- **N6.FRER. Out of order frames:** An out of order FRER frame might indicate a malicious packet injection or a faulty endpoint. This notice alerts any out of order frame and if it should be dropped following the same mechanism as the stream recovery function in the corresponding TSN bridges.
- **N7.FRER. Excessive member streams:** FRER duplicates member streams based on the configured degree of redundancy. This notice is raised if the number of received duplicate packets for a stream is more than the degree of redundancy.
- **N8.FRER. Terminated member streams:** This notice alerts if a member stream is not active, as it may indicate a node or link failure as well as an attack.

B. Intrusion Detection Component

The intrusion detection component consists of the detection engine and another broker to communicate with the monitoring module. We implemented the detection engine purely in Python v3.9.2. It performs traffic analysis to (i) keep the current states of different streams and their configurations, (ii) make per-frame or periodical examinations to detect potential anomalies. Accordingly, it publishes the respective alerts via the broker to be logged by the notice engine. Note that while the first broker (attached to the monitoring component in Fig. 1) disseminates the frames from the event engine to others, this one establishes the communication between the notice and detection engines. It enables us to design the intrusion detection component as a standalone module that can be replaced by any other intrusion detection logic.

The detection engine in this component introduces a set of functions to detect various SRP and FRER threats listed in [1]. These functions are analogous to the rules in a rule-based IDS. Therefore, they are extendable to detect further threats simply as adding new rules. In the remaining of this section, we describe the detection functions together with the attacks they can recognize. We also note on the alternative placements of `TSNZeek` in the network, i.e., centralized, local, or peripheral in the network to detect the described attacks.

- **A1.SRP. Unusual SRP request:** An attacker can send malicious SRP requests to a CNC or an edge TSN bridge such as (a) demanding a bulky network bandwidth for a stream or (b) registering several streams to exhaust available resources. The detection engine detects such scenarios by comparing the requested stream traffic specifications extracted from the `TrafficSpecification` header of SRP frames with predefined threshold values for the maximum bandwidth and frame rates. It also keeps the rolling average of those values to recognize if an attacker requests stream reservations whose traffic characteristics significantly differ from the average.
- **A2.SRP. Flooding SRP requests:** An attacker can flood SRP requests to exhaust available resources quickly. The detection engine limits the rate of incoming requests and alerts for excessive requests. The rate limit is predefined and configured by the administrator.
- **A3.SRP. Changing existing allocation:** An attacker can forge an SRP request for an already registered stream to (i) reduce its reserved resources to degrade its service quality or (ii) increase its reserved resources to exhaust available resources without injecting any new stream that could be easily recognizable otherwise. The detection engine can automatically deduce if a request is accepted by checking the `TalkerStatus` group header of an SRP response. Then, for each SRP request, it checks if there already exists an accepted stream and alerts for one of the scenarios above.
- **A4.SRP. Dangling resources:** An attacker can reserve network resources to manipulate resource utilization without sending any real data traffic since it can also be detected and filtered by firewalls or network policies. For such cases, the detection engine periodically checks if the reserved resources are in-use. It alerts for the streams without any processed frames within a predefined time threshold.

`TSNZeek` should monitor all SRP traffic to detect the listed SRP-specific attacks. Therefore, while a centralized SRP configuration (see Section II-A) imposes a centralized `TSNZeek` deployment, a distributed one requires monitoring edge bridges.

- **A5.FRER. Forging fake sequence numbers:** If an attacker can observe the current sequence number of a FRER stream, it can inject malicious frames with that sequence number so that the legit frame would be dropped by the sequence recovery function in TSN bridges. If the attacker injects a frame with the upcoming sequence number, the detection engine alerts when it detects more than one frame with the same sequence number. Besides, it deduces the expected degree of redundancy, i.e., the number of expected duplicate frames for a stream, by processing the `NumSeamlessTrees` header in the `UserToNetworkRequirements` group header of the SRP request during registration of the respective stream. If the attacker searches for the legit sequence number by sending frames with the random sequence numbers, the detection engine raises an alert for an out of order frame.
The detection engine mimics the stream recovery function of FRER to keep track of the legitimate intervals of the expected sequence numbers. Thus, it needs to be configured with the same recovery function, match or vector recovery used by the TSN bridges in the system. This also requires monitoring TSN bridges locally to detect where exactly malicious frames are injected and dropped.
- **A6.FRER. Malicious rerouting:** If there are intersections between redundant paths, it eliminates duplicate packets being forwarded through the same bridge [16]. Instead of directly sabotaging the communication, an attacker could subtly reroute the redundant stream through intersecting paths to force FRER to drop the duplicate packets and hinder the redundancy. Therefore, `TSNZeek` examines the configured FRER routes, e.g., against malicious misroutings, intersecting paths etc. This requires the `TSNZeek` attached to the SRP controller,

TABLE I: The overview of the notices, attack detection functions, and other aspects of the intrusion detection component.

Protocol	Detection	Notices	Frequency		Deployment			Context	
			Per-frame	Period.	Central.	Local	Edge	Manual	Stateful
SRP	A1.SRP	N1.SRP, N2.SRP	✓	–	✓	–	✓	Resource threshold	Reservations
	A2.SRP	N1.SRP, N2.SRP, N3.SRP	✓	–	✓	–	✓	Rate-limit	–
	A3.SRP	N1.SRP, N2.SRP, N4.SRP	✓	–	✓	–	✓	–	Reservations
	A4.SRP	N5.SRP		✓	✓	✓	✓	–	Reservations
FRER	A5.FRER	N6.FRER, N7.FRER	✓	–	–	✓	–	–	Seq. numbers
	A6.FRER	N7.FRER	✓	–	–	✓	✓	–	Seq. numbers
	A7.FRER	N7.FRER, N8.FRER	✓	✓	–	–	✓	Timeout	Seq. numbers

i.e., centralized or hybrid, to access to the configuration of redundant paths.

• **A7.FRER. Triggering timeout:** An attacker can enforce FRER functions on TSN bridges to raise a `RECOVERY_TIMEOUT` event [7] if it can block all member streams of a FRER stream. Consequently, the expected sequence number of that stream is revoked. Once the attacker sends the first frame after this event, it becomes the valid originator of the stream with the forged initial sequence number. The detection engine recognizes the absence of the original member streams by measuring the time passed after the reception of the last frame of the respective streams. It also detects if the same stream has a new sequence number by per-frame examinations. Both can be detected by monitoring the edge bridge that the destination endpoint is attached to.

Table I summarizes attack detection and notices with several related aspects discussed above. These aspects are (i) how frequently `TSNZeek` investigates an attack, i.e., per-frame or periodically, (ii) how `TSNZeek` instances should be deployed to detect an attack, i.e., centralized, local, or to the edge, and (iii) what `TSNZeek` needs for the detection, i.e., manual configuration and the current state of stream reservations.

V. Evaluation

This section presents our experimental setup and evaluation results for the efficiency and performance of `TSNZeek`.

A. *Experimental Setup*

For our experiments, we set up a real TSN testbed shown in Fig. 4. It consists of three TSN bridges (TSN1, TSN2, and TSN3) connected in a ring topology. An endpoint (EP1) is attached to TSN3, and another (EP2) is attached to TSN2. `TSNZeek` is deployed on a computer with an Intel Core i3-9100 3.60Ghz CPU and connected to TSN1. A malicious endpoint (MEP) is also attached to TSN1 to conduct attacks. In our application scenario, EP1 first sends an SRP request for a resource reservation to stream a video. Then, it sends the data over two redundant paths, TSN2-TSN1-TSN3 and TSN2-TSN3, to EP2 and EP3 using FRER. `TSNZeek` monitors FRER frames forwarded over TSN1.

Since there is not any public TSN dataset including malicious traffic, we also implemented attacks described in Section IV-B in Python (available in the source code).

B. *Results*

We evaluated the resource usage and intrusion detection capabilities of `TSNZeek`. For resource usage, we measured the CPU utilization of the monitoring and detection components, as well as the packet processing rate and delay of `TSNZeek`. For intrusion detection, the typical evaluation metrics for an IDS, e.g., accuracy, sensitivity [17], are not directly fitting for our rule-based IDS as its objective is detecting the specific threats according to the implemented rules. Therefore, we tested the effectiveness of the detection module against the attacks described in Section IV-B.

1) Resource usage: Since IEEE 802.1 TSN protocols define data link layer protocols, processing the events starting from such low-level communication may easily lead to high resource usage. Accordingly, we measured the CPU usage of `TSNZeek` for an increasing data load from 50 to 250 Mbit/s. This interval of data load is reasonable for the number of critical streams in a TSN network and mainly limited by the processing capacity of the TSN bridges in our testbed. We generated the load using *iperf*, which is a network speed test tool. Fig. 5 shows the resource usage of the monitoring component (Zeek module extended with TSN grammar, red and solid line), the detection component (Python program running at the control plane, blue and dashed line), and also the CPU consumption of native Zeek without TSN support (processing non-TSN Ethernet frames, black and dotted line).

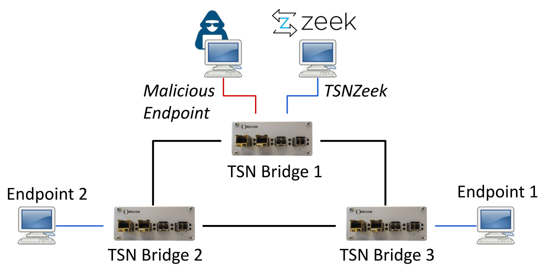


Fig. 4: Testbed setup.

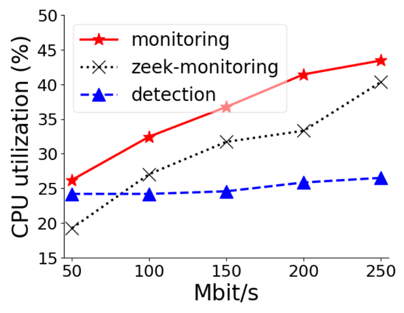


Fig. 5: CPU utilization of the monitoring and detection components.

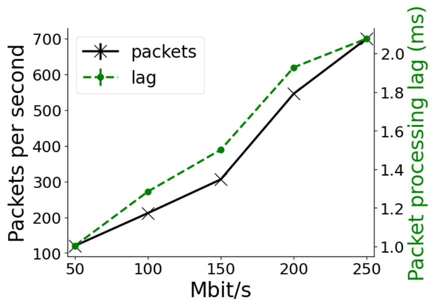


Fig. 6: Packet processing performance of the monitoring component.

When increasing the traffic load, the CPU utilization of the monitoring component increases from 25% at 50Mbit/s to 45% at 250Mbit/s. As shown in the figure, `TSNZeek` consumes only ∼5% more CPU power than the Zeek instance without TSN support. The detection module has a constant resource utilization of around 25% as it only processes singular events that are sent by the monitoring module.

Fig. 6 shows the packet processing rate and lag of `TSNZeek`. The packet processing lag describes the time passed between the reception and parsing of a frame. The figure shows that the packet processing rate (black, solid line) increases proportionally with an increasing data load without any packet drops, e.g., due to a potential congestion. Increasing load also leads to a higher packet processing lag of up to 2 ms (green, dashed line). For any lag in milliseconds, time-sensitive frames with submillisecond latency requirements may already be delivered before an intrusion alert. Although it is not critical for a monitoring module, a potential time-sensitive intrusion *prevention* system utilizing `TSNZeek` might require further improvements in data processing speed.

```
"date": "2022–10–26–16–41–02",
"timestamp": 1666802462.036670,
"note": "TSN::POTENTIAL_ATTACK_6",
"protocol": "FRER",
"msg": "Out of order frame is discarded for stream 29695 – seq.nr.
    7148 < 54972",
"actions": "Notice::ACTION_LOG"
```

Fig. 7: A sample intrusion alert in *json* format.

2) Intrusion detection: In our experiments, `TSNZeek` can successfully detect all the listed attacks in Section IV-B and raise the respective notices in real-time. Fig. 7 shows N6.FRER (in json format) in the log stream of the notice engine against the attack A5.FRER, i.e., the frame injection with the sequence number 7148, while the expected one is 54972.

However, we still observe redundant notifications in particular scenarios. For instance, when we connect an Ethernet hub between TSN1 and TSN3, e.g., extending the network with a non-TSN network component, a member stream delivers out of order packets due to the delayed frames. `TSNZeek` notices this as a malicious attempt because of highly deviating sequence numbers. Although this is unusual for strictly configured TSN systems, `TSNZeek` should still be configured considering such network conditions.

VI. Conclusion

Although IEEE 802.1 TSN standards propose emerging time-sensitive communication protocols for critical systems, they still lack security countermeasures against potential attack vectors. In this paper, we present the first open-source security monitoring and intrusion detection system, `TSNZeek`, for IEEE 802.1 TSN protocols. We implement `TSNZeek` by extending an existing monitoring tool, Zeek, with a new packet parsing grammar and the stream analysis engines addressing TSN-specific security events and attacks. We evaluate its resource usage and confirm that it can successfully detect various attacks against the prominent TSN protocols, SRP and FRER. For future work, we aim to improve our detection engine to minimize redundant and false alerts by accurately modeling the usual TSN behavior.

References

[1] D. Ergenç, C. Brülhart, J. Neumann, L. Krüger, and M. Fischer, “On the Security of IEEE 802.1 Time-Sensitive Networking,” in *IEEE Int. Conf. on Communications Workshops (ICC Workshops)*, 2021.

[2] P. Meyer, T. Häckel, S. Reider, F. Korf, and T. C. Schmidt, “Network Anomaly Detection in Cars: A Case for Time-Sensitive Stream Filtering and Policing,” *CoRR*, vol. abs/2112.11109, 2021.

[3] P. Meyer, T. Häckel, F. Korf, and T. C. Schmidt, “DoS Protection through Credit Based Metering - Simulation Based Evaluation for Time-Sensitive Networking in Cars,” *CoRR*, vol. abs/1908.09646, 2019.

[4] IEEE 802.1 TSN Task Group, “IEEE Std. for Local and Metro. Area Net. – Bridges and Bridged Networks – Amendment 28: Per-Stream Filtering and Policing,” *IEEE Std 802.1Qci-2017*, 2017.

[5] V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time,” in *7th USENIX Security Symposium*, Jan. 1998.

[6] IEEE 802.1 TSN Task Group, “IEEE Std. for Local and Metro. Area Net. – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements,” *IEEE Std 802.1Qcc-2018*, 2018.

[7] IEEE 802.1 TSN Task Group, “IEEE Std. for Local and Metro. Area Net. – Frame Replication and Elimination for Reliability,” *IEEE 802.1CB-2017*, 2017.

[8] F. Fischer and D. Merli, “Security considerations for ieee 802.1 time-sensitive networking in converged industrial networks,” in *Int. Conf. on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pp. 1–7, 2022.

[9] M. Topsakal and S. Cevher, “Impact Analysis of Denial of Service Attacks in IEEE 802.1 Time Sensitive Networking,” in *30th Signal Processing and Communications Applications Conference (SIU)*, 2022.

[10] P. Meyer, T. Häckel, F. Korf, and T. C. Schmidt, “Network Anomaly Detection in Cars based on Time-Sensitive Ingress Control,” in *IEEE 92nd Vehicular Technology Conference*, 2020.

[11] F. Luo, B. Wang, Z. Fang, Z. Yang, Y. Jiang, and K. Demertzis, “Security Analysis of the TSN Backbone Arch. and Anomaly Det. System Design Based on IEEE 802.1Qci,” *Sec. and Comm. Networks*, 2021.

[12] R. Barton, M. Seewald, and J. Henry, “Management of IEEE 802.1Qci Security Policies for Time Sensitive Networks (TSN),” tech. rep., Technical Disclosure Commons, 10 2018.

[13] T. Bu, Y. Yang, X. Yang, W. Quan, and Z. Sun, “TSN-Insight: An Efficient Network Monitor for TSN Networks,” APNet, 2019.

[14] L. Muguira, J. Lázaro, S. Alonso, A. Astarloa, and M. Rodriguez, “Secure Critical Traffic of the Electric Sector over Time-Sensitive Networking,” in *IEEE 35th Conference on Design of Circuits and Integrated Systems (DCIS)*, 2020.

[15] IEEE 802.1 TSN Task Group, “IEEE Std. for Local and Metro. Area Net. – Media Access Control (MAC) Security,” *IEEE Std. 802.1AE-2018*, 2018.

[16] D. Ergenç and M. Fischer, “On the Reliability of IEEE 802.1CB FRER,” in *IEEE Int. Conf. on Computer Communications (INFOCOM)*, 2021.

[17] L. N. Tidjon, M. Frappier, and A. Mammar, “Intrusion Detection Systems: A Cross-Domain Overview,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3639–3681, 2019.

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ort, Datum	Unterschrift

Ich bin damit einverstanden, dass meine Arbeit in den Bestand der Bibliothek eingestellt wird.

Ort, Datum	Unterschrift