Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Cumulative Doctoral Dissertation

Towards Flexible and Resilient Next Generation Time Sensitive Networks

submitted by
Nurefşan Sertbaş Bülbül

Dissertation to receive the title *Dr. rer nat.*

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

Hamburg, 09.10.2023

Reviewers

Prof. Dr. Mathias Fischer
Prof. Dr. Andreas J. Kassler

Disputation date: 21.12.2023

Abstract

Mission-critical communication requires delivering data in real-time, reliably, and securely. Over the past few years, mission-critical networks have become more complex due to cheap commercial off-the-shelf equipment and increasing convergence with other networks like the Internet. They have to cope with more diverse and dynamic traffic with ultra-high reliability and various quality of service (QoS) requirements. Time-sensitive networking (TSN) has been introduced to address these issues by unifying diverse networking equipment and protocols to facilitate the transmission of convergent data flows over cost-effective and easily deployable standard Ethernet technologies. Moreover, the redundancy mechanisms offered by TSN enhance the network reliability and ensure seamless communication during single link or node failures. However, configuring these networks and ensuring resilience against attacks while maintaining the desired QoS remains challenging and thus needs further investigation.

Accordingly, this *cumulative* thesis presents contributions to support the self-adaptive and efficient configuration of time-sensitive networks and methods for increasing their resilience against denial of service (DoS) attacks.

To bring self-adaptation to time-sensitive networks and make them more efficient, we present a framework that makes them fully transparent to end hosts and eliminates the need for their involvement. Furthermore, we propose reconfiguration heuristics to improve the network utilization and to accommodate more flows in the network without sacrificing the QoS. Lastly, we explore the potential of machine learning as a low-cost alternative to optimize flow reservations within time-sensitive networks.

To increase the resilience of time-sensitive networks against DoS attacks and maintain the QoS, we propose different admission control strategies via in-network filtering to enforce flow reservations. Moreover, we investigate the impacts of calibrated attacks in time-sensitive networks and discuss a few countermeasures to prevent them. Lastly, we present a distributed denial of service (DDoS) attack detection and collaborative filtering approach that operates at the network layer and protects time-sensitive hybrid networks that carry time-triggered and best-effort traffic at the same time.

By integrating the flexible configuration strategies and resilience aspects presented, we contribute to advancing next-generation time-sensitive networks. While these enhancements are significant steps forward, they do not address all challenges in the TSN domain, indicating that further research and development efforts remain essential.

Zusammenfassung

Bei der unternehmenskritischen Kommunikation müssen Daten in Echtzeit, zuverlässig und schnell übermittelt werden. In den letzten Jahren sind unternehmenskritische Netzwerke aufgrund günstiger kommerzieller Standardgeräte und der zunehmenden Konvergenz mit anderen Netzwerken wie dem Internet immer komplexer geworden. Sie müssen einen vielfältigeren und dynamischeren Datenverkehr mit besonders hoher Zuverlässigkeit und verschiedenen Anforderungen an die Dienstqualität (QoS) bewältigen. Time Sensitive Networking (TSN) wurde eingeführt, um diese Probleme zu bewältigen, indem es verschiedene Netzwerkkomponenten und -protokolle vereinheitlicht, um die Übertragung von konvergenten Datenströmen über kostengünstige und leicht zu implementierende Standard-Ethernet-Technologien zu erleichtern. Darüber hinaus erhöhen die von TSN angebotenen Redundanzmechanismen die Zuverlässigkeit des Netzwerkes und stellen eine nahtlose Kommunikation bei Ausfällen einzelner Verbindungen oder Knoten sicher. Der Aufbau dieser Netzwerke und die Sicherstellung der Widerstandsfähigkeit gegen Angriffe bei gleichzeitiger Aufrechterhaltung der gewünschten Dienstgüte bleibt jedoch eine Herausforderung und bedarf daher weiterer Untersuchungen.

Dementsprechend werden in dieser kumulativen Dissertation Beiträge zur Unterstützung der selbstanpassenden und effizienten Konfiguration zeitempfindlicher Netzwerke und Methoden zur Erhöhung ihrer Widerstandsfähigkeit gegen Denial-of-Service-Angriffe (DoS) vorgestellt.

Um die Selbstanpassung in zeitkritischen Netzwerken zu ermöglichen und sie effizienter zu machen, stellen wir ein Framework vor, der sie für die Endhosts völlig transparent macht und ihre Beteiligung überflüssig macht. Darüber hinaus schlagen wir Rekonfigurationsheuristiken vor, um die Netzwerkauslastung zu verbessern und mehr Datenströme im Netzwerk unterzubringen, ohne die Dienstgüte zu beeinträchtigen. Anschließend untersuchen wir, wie maschinelles Lernen als kostengünstige Alternative zur Optimierung von Flussreservierungen in zeitabhängigen Netzwerken vorteilhaft eingesetzt werden kann.

Um die Widerstandsfähigkeit zeitempfindlicher Netze gegen DoS-Angriffe zu erhöhen und die Dienstgüte aufrechtzuerhalten, schlagen wir verschiedene Zulassungskontrollstrategien mittels netzinterner Filterung vor, um Flussreservierungen durchzusetzen. Darüber hinaus untersuchen wir die Auswirkungen kalibrierter Angriffe in zeitkritischen Netzwerken und diskutieren einige Gegenmaßnahmen, um diese zu verhindern. Schließlich stellen wir einen Ansatz zur Erkennung von verteilten Denial-of-Service-Angriffen (DDoS) und zur kollaborativen Filterung vor, der auf der Netzwerkebene arbeitet und zeitempfindliche hybride Netzwerke schützt, die gleichzeitig zeitgesteuerten und Best-Effort-Verkehr übertragen.

Durch die Integration der vorgestellten flexiblen Konfigurationsstrategien und Ausfallsicherheitsaspekte tragen wir dazu bei, zeitempfindliche Netzwerke der nächsten Generation voranzubringen. Obwohl diese Verbesserungen einen bedeutenden Fortschritt darstellen, können sie nicht alle Herausforderungen im TSN-Bereich bewältigen, was zeigt, dass weitere Forschung und Entwicklung auf diesem Bereich weiterhin nötig sein wird.

Acknowledgement

I would like to begin this section by acknowledging the immense journey that pursuing a PhD entails. The path has long been filled with challenges, growth, and unwavering dedication. This is not merely an academic accomplishment; it represents years of relentless pursuit of knowledge, countless hours of research and analysis, and a profound commitment to pushing the boundaries of human understanding. Throughout this journey, I have been fortunate to receive support, guidance, and inspiration from many individuals, without which this achievement would not have been possible.

First and foremost, I would like to thank my supervisor, Prof. Mathias Fischer, as I will stay grateful for his guidance and support provided throughout my PhD journey. His constant commitment to discipline and dedication to the pursuit of knowledge served as a constant source of inspiration. As a female researcher, I deeply appreciate his consistent encouragement and support, which played a pivotal role in my academic and personal growth.

I would also like to express my appreciation to my research group as a whole. Your collaborative spirit have been instrumental in my academic growth. I want to express my gratitude to Malte Hamann for his warm welcome when I first arrived in a new country and university. I am deeply thankful for the support and friendship of Florian Wilkens. His humor has brightened even the most challenging academic tasks. Tatjana Wingarz is a cherished friend, and our shared interests and personalities have made our interactions feel like looking into a mirror. It was a chance to work with Cornelia Brülhart; her cheerful disposition not only smoothed our professional interactions but also turned each encounter into a delightful experience. I must also thank Mathias again for hiring Doğanalp Ergenç and enabling our fruitful partnership. We've not only worked together but also our friendship extended beyond our workplace with a few conference trips together.

Finally, my family holds a special place in my heart, as they have consistently been my pillars of strength and constant supporters throughout my life's journey. They have never hesitated to stand by my side, wholeheartedly endorsing my decisions and dreams. I want to thank my life partner, Şeref, for this unyielding belief in my potential, which has been a constant source of motivation, propelling me to pursue my goals. I am profoundly grateful for the love, guidance, and support he provides, which continue to shape my path and inspire me to reach new heights. Lastly, I want to extend my deepest gratitude to my unborn child, who has already brought an immense sense of joy and purpose into my life. You are a source of boundless hope, and I am excited to embark on this new chapter of life with you by my side.

TO MY PARENTS
for raising me to believe that anything was possible
AND TO MY HUSBAND
for making everything possible

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Problem Statement

Mission-critical networks (MCNs) are essential for the efficient and resilient operation of critical systems, such as industrial networks, transportation systems, and emergency services. They are required to be highly reliable, often incorporating redundancy and failover mechanisms to ensure availability. Besides, these networks have strict quality of service (QoS) criteria, e.g., low and bounded latency. However, ensuring a high degree of reliability while adhering to these strict QoS requirements is a challenging task when using traditional networking technologies. This is where IEEE 802.1 time-sensitive networking (TSN) comes in as a solution. It offers deterministic, low-latency communication over Ethernet networks and is standardized by the IEEE task group [IEE17a]. TSN ensures the prioritization and dependable delivery of critical traffic while allowing different traffic classes with different QoS requirements to coexist on the same network. Thus, it supports and enables efficient and resilient operation of critical infrastructures.

TSN has a wide variety of application scenarios that are defined by a set of IEEE standards e.g., industrial automation. The convergence of information technology (IT) and operations technology (OT) in automotive and industrial automation environments requires effective network management solutions that can facilitate real-time communication and guarantee optimal levels of QoS for different criticality traffic. TSN closes this gap by presenting different network management and configuration mechanisms. Figure 1.1 shows a typical TSN use case, namely an in-factory network. Each machine or device in the field layer is connected to TSN switches[1] via Ethernet, allowing them to communicate in real-time. For instance, the temperature sensor must measure the temperature regularly to ensure the safe operation of devices like motors that have to operate within a specific temperature range. If the temperature exceeds the safety limit, this data must be sent to the control system in real-time, as it might be needed to slow down or shut down the motor. Large communication delays result in motor damage, leading to production delays and safety hazards. TSN can ensure that critical data is prioritized and delivered with low latency and high reliability. On higher levels, the TSN

[1] Throughout this thesis, the term *flow* is used interchangeably with *stream*, and *switch* is used interchangeably with *bridge* in the context of TSN terminology, as elaborated in Section 2.1.

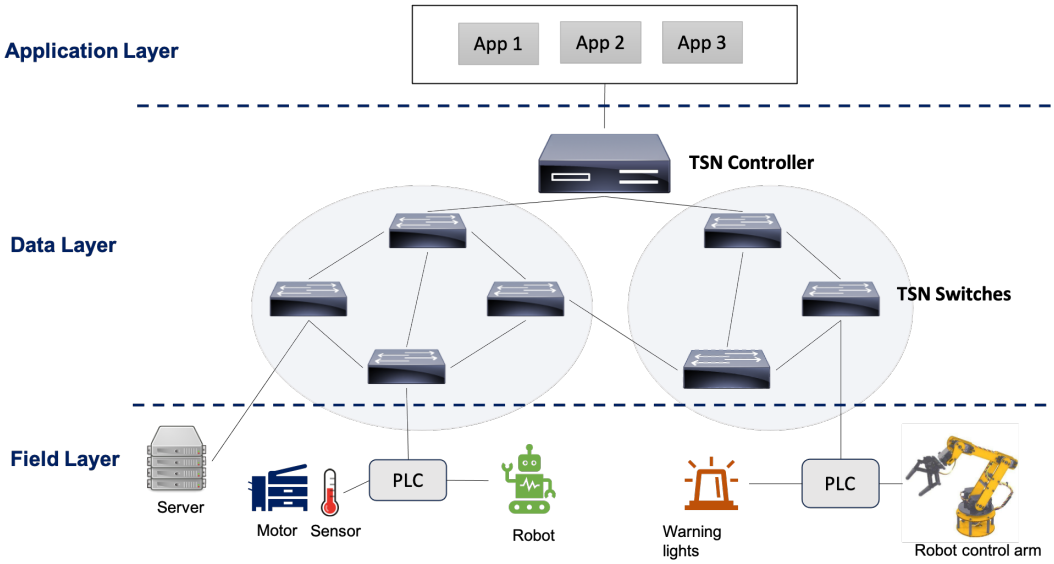


Figure 1.1: Industrial time-sensitive network scenario.

controller is logically centralized; i.e., it has a global view in topology and traffic, which is required to realize network control logic such as routing and flow scheduling. At the application layer, several applications run on top of the controller and can implement more specific functionality, e.g., traffic monitoring, QoS management.

While TSN is evolving towards new use cases with increasing dynamicity and heterogeneity in network elements and in the traffic, and complexity of configuration, ensuring the network's resilience against attacks to maintain the desired QoS becomes harder. This poses the following major problems:

Problem 1: Traditional time-sensitive networks typically rely on a prior fixed configuration scheme that requires the active participation of TSN-aware static network entities. In such networks, paths for data flows are usually established at the startup time of an application and remain as it is during the lifetime of the flow. This limits the adaptability of the network in the case of changing flow resource requirements. Furthermore, nowadays, mobile assets are increasingly deployed in industrial environments. For instance, mobile robots can perform tasks like transporting goods as automated guided vehicles (AGV) or assisting in manufacturing processes [Gin21]. This mobility leads to even more dynamics; devices often change their place thus, flows are added or removed from the network more frequently, resulting in fragmented link bandwidths. This requires more dynamic traffic management solutions and increases the complexity as this brings updates of many configuration parameters.

Problem 2: In the future, it is expected that time-sensitive networks will serve as a backbone of modern communication systems require strict resiliency measures. Refer-

ring to the example scenario illustrated in Figure 1.1, an external attacker gaining access can launch a cyber attack like denial of service (DoS), specifically targeting the TSN infrastructure. By flooding the network with an overwhelming traffic volume, the attacker disrupts the precise synchronization of machines and robots or even shuts down the entire production line. Moreover, latency, a unique concern in this environment, is critical for performance and security. Unexpected delays or interruptions can breach strict QoS requirements and disrupt production. However, security was not the primary focus during Ethernet network design and also of TSN. Therefore, addressing threats specific to TSN mechanisms is essential for ensuring the resilient operation of these networks and maintaining continuous and predictable network performance.

Hence, the increasing complexity arising from diverse QoS demands and increasing dynamicity requires additional network management strategies. While these strategies should facilitate the network configuration and orchestration, they must also ensure resilience and meet QoS criteria.

1.2 Research Questions

TSN is a promising solution for future MCNs, but it introduces significant configuration and resilience issues. In the remainder of this section, we[2] first introduce research questions regarding the **self-adaptive and efficient configuration** of time-sensitive networks and then research questions regarding the **resilience against denial of service attacks** within these networks.

Although several TSN standards have been established, configuring these networks is still challenging. This is mainly because of the significant number of configuration parameters that are jointly configured by TSN-aware end hosts and TSN switches. Moreover, the dynamic nature of these networks, such as changes in traffic patterns, needed to be signaled to the network to adapt to such changing conditions. Thus, the main research question that emerges from this context is as follows:

> RQ1: How to design an autonomous configuration solution for seamless TSN resource reservation without requiring end hosts to be TSN-aware, while ensuring the desired QoS and reducing the configuration overhead?

Following the self-configuration of time-sensitive networks, it has become significant to efficiently utilize the network resources, such as bandwidth, as the number of flows increases during the network lifetime. In standard TSN deployments, routing paths are typically configured at the startup time of an application and remain fixed. This

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

can lead to suboptimal assignments of flows as they cannot adapt to changing network conditions, and some links/switches may be underutilized while others may become overloaded. As a result, a new research question arises as follows:

> RQ2: How to dynamically reconfigure time-sensitive networks at runtime to efficiently use network resources considering reconfiguration overhead? Is it feasible to do this in real-time?

To configure time-sensitive networks while considering both limited network resources and individual end-to-end latency requirements, optimization problems can be employed to determine the optimal routing paths. The optimization problems seek solutions for the given objective by ensuring all constraints are met. However, varying QoS requirements of internet of things (IoT) applications and larger network topologies increase the computational complexity exponentially. Thus, machine learning (ML) driven resource reservation strategies became promising alternatives for optimization problems. Accordingly, the main research question that emerges from this context is as follows:

> RQ3: How can ML-driven strategies be leveraged to achieve efficient resource reservations in TSN? How close can these strategies approximate optimal performance?

These research questions towards a self-adaptive and efficient configuration of time-sensitive networks consider dynamic environmental conditions (RQ1), more efficient resource distribution through reconfiguration (RQ2), and low-cost, scalable resource allocations with the help of ML (RQ3).

In addition to the management and organization of TSN, the resilient operation of these networks is essential. Any malicious attempts in such systems can have significant consequences, leading to serious financial, operational, and safety risks. The need for reliable measures becomes increasingly apparent as more industries adopt TSN technologies. Therefore, in the subsequent sections, we will establish several research questions related to resilience against DoSattacks, specifically in time-sensitive networks.

Time-sensitive networks are precisely calibrated to ensure that delays in the network are deterministic. However, this predictability can make TSN susceptible to malicious elements, such as compromised talkers or switches that behave in an entirely unanticipated manner. Thus, detecting such behaviors and mitigating these malicious attempts immediately is important for legitimate time-sensitive traffic. Here, the following research question arises:

> RQ4: What types of admission control strategies could effectively enforce flow reservations, and what would be the associated deployment costs?

There are different TSN mechanisms for guaranteeing bounded latency depending on the application and/or traffic requirements. Given that these mechanisms exhibit deterministic behavior, attackers can capture traffic characteristics. They can use this information to target the network by specific DoS attacks to degrade the QoS of TSN. Therefore, the main research question here is:

> RQ5: What information can an attacker gain from passive monitoring of TSN traffic, and is this information exploited for attacking time-sensitive networks? Moreover, what countermeasures can be used to protect the network against such malicious activities?

Not only does the violation of resource reservation limits lead to a degradation of QoS, but collaborative attackers can also instigate a DoS. This can be even more challenging to detect since the end hosts adhere to their limits, making them appear completely legitimate. Thus, detecting such behaviors and mitigating these malicious attempts immediately is important for TSN to maintain QoS. Here, the following research question arises:

> RQ6: How can malicious and legitimate traffic distinguished, and how this be leveraged to eliminate or minimize the impact of attacks in near-real-time?

These research questions address the resilient operation of time-sensitive networks via dynamic admission control strategies (RQ4), impacts and countermeasures of calibrated attacks (RQ5), as well as dynamic DoS attack detection and mitigation mechanisms (RQ6).

1.3 Contributions

This section answers the stated research questions regarding flexible and resilient time-sensitive networks and summarizes the contributions of this thesis. The contributions are grouped according to the two primary objectives of this thesis: (i) ensuring self-adaptive and efficient configuration and (ii) ensuring resilience against DoS attacks for time-sensitive networks. The following sections highlight our key findings, represented in Figure 1.2. Here, we presume one generic TSN deployment is based on a central network configuration. Then, we illustrate our developed mechanisms per contribution and their possible placement in the network accordingly.

Figure 1.2: The main contributions of this thesis using a centralized configuration architecture as the foundation for TSN.

Self-Adaptive and Efficient Configuration of Time Sensitive Networks

This section overviews our contributions regarding the self-adaptive and efficient configuration of time-sensitive networks. Within this context, we will present (i) a dynamic self-configuration framework for removing the end-host-related dependencies, (ii) reconfiguration strategies for better resource utilization, and (iii) machine learning-based intelligent configuration for TSN.

C1: Dynamic Self-configuration of Time Sensitive Networks
This contribution handles the dynamic self-configuration of time-sensitive communication. Here, we benefit from the software-defined networking (SDN) concept that separates the control plane from the data plane, allowing centralized software to dynamically manage and configure network resources, enhancing flexibility and programmability. We propose a self-configuration framework for TSN in [SEF21] that follows the plug-and-play nature of Ethernet networks and can help to address RQ1. The framework removes the end-host-related dependencies of TSN. Flows are initially placed on default paths to extract traffic characteristics before migrating them to better or even optimal paths. To obtain the traffic characteristics, we monitor incoming network traffic at edge switches, and we use these characteristics to move flows to optimal paths while maintaining hard real-time guarantees. For that, we formulated an optimization problem. Our solution can automatically extract traffic resource requirements and is completely transparent to the end hosts.

C2: Reconfiguration Strategies for Time Sensitive Networks
This contribution addresses the challenge of dynamically reconfiguring the path of time-sensitive flows while adhering to strict latency constraints. To address research question RQ2, in [SEF22], we propose flow placement heuristics that add distinct constraints to the optimization problem presented in our previous work [SEF21]. Additionally, we leverage concepts from SDN, such as global network view in near-real-time, to gather and integrate application requirements into the network resource configuration process. Our heuristics offer reconfiguration solutions for different environments, and we assert that by considering the reconfiguration overhead and the characteristics of the environment, an appropriate reconfiguration strategy can be chosen.

C3: Machine-Learning-based Intelligent Configuration of Time Sensitive Networks
This contribution dynamically handles the resource reservation for time-sensitive networks. For that, we propose an approach based on reinforcement learning (RL) that interacts with the network environment and learns the optimal resource allocations, e.g., routing paths [SF22]. With this, we can address research question RQ3. The objective, a reward function in the RL approach, is thoughtfully designed to ensure that individual

flow deadlines are met. A straightforward solution to this resource reservation problem is formulating a mixed integer linear programming (MILP), known to be NP-hard and thus computationally complex. Thus, our proposed RL-based approach stands as a low-cost alternative solution.

Resilience against Denial of Service Attacks for Time Sensitive Networks

This section overviews our contributions regarding the resilient operation of time-sensitive networks, which ensures the transmission of time-sensitive data within strict time constraints, preventing potential disruptions that could harm critical operations. Accordingly, we will present (i) network admission control strategies specifically designed for TSN, (ii) calibrated attacks to TSN frame preemption mechanism and countermeasures, and (iii) dynamic and scalable DoS attack detection and collaborative filtering approaches.

`C4`: Admission Control Strategies for Time Sensitive Networks
This contribution handles the vulnerabilities of the standard resource reservation mechanism of the TSN, which is susceptible to traffic overload and DoS attacks. To address the research question `RQ4`, we propose a programming protocol-independent packet processors (P4)-based dynamic in-network attack filtering solution in [SKF23], to defend the network against compromised or faulty network elements such as talkers or switches directly on the data plane.

`C5`: Calibrated Attacks against TSN Frame Preemption and Countermeasures
This contribution focuses on calibrated attacks against TSN frame preemption. To address the research question `RQ5`, various potential attack scenarios are described, and their potential impacts on network performance are evaluated in [SF23]. Furthermore, a methodology is presented to demonstrate the feasibility of these scenarios by utilizing passive observation of TSN traffic by the attackers and degrading the QoS in the network. Lastly, we also discuss a few countermeasures to avoid these attacks.

`C6`: Dynamic and Scalable DoS Attack Detection and Filtering
This contribution focuses on handling more larger-scale collaborative attacks. To address the research question `RQ6`, we employ attribute-oriented induction (AOI)-based attack signature extraction strategy for deriving filtering rules with up-to-date data sets containing different types of DoS attacks. Then, we collaboratively filter attacks close to its source with the help of SDN and network function virtualization (NFV) concepts in [SF20]. Even though TSN is not used as an environment in this study, these filtering strategies can be directly coupled with time-sensitive networks to mitigate DoS attacks.

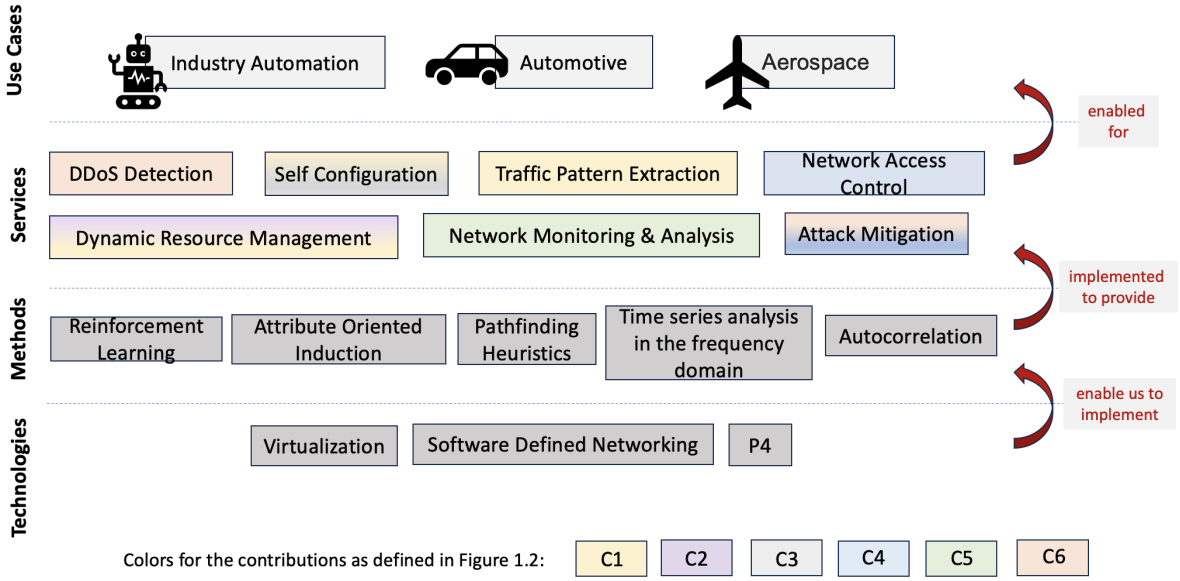


Figure 1.3: An overview of developed services and used enabler technologies regarding individual thesis contributions.

A comprehensive representation of our contributions regarding various technologies and methods to provide different services for varied use cases is presented in Figure 1.3. We use different technologies that enable us to implement different methods, e.g., reinforcement learning and heuristics. These methods provide different network services, such as DDoS detection and mitigation. The figure also includes typical use cases that could benefit significantly from the provided services within the context of this thesis.

Further Contributions

In addition to the contributions mentioned above, we developed further solutions for designing next-generation networks. In our previous work, [WBC+20], we introduce a virtualized testbed that can be seamlessly integrated with actual automation hardware. This approach eliminates the requirement for costly dedicated hardware setups for security testing and enables the evaluation of security strategies with greater flexibility. In [SEF22], we summarize key insights and outline potential research directions for enhancing the reconfigurability of time-sensitive flows.

Furthermore, we highlight the importance of TSN protocols for mission-critical systems and the need for advanced methods, particularly artificial intelligence (AI)/ML models, to design and maintain time-sensitive networks within these systems. In [ESM+23], we discuss and emphasize the necessity for TSN datasets to facilitate research on AI/ML based techniques for TSN systems and explore alternative designs to create realistic datasets. Moreover, in [SF23], we propose to use a reinforcement learning-based adaptive scheduling approach to utilize network resources more efficiently and increase the

Chapter	Section	Contribution	Publications
Chapter 1: Introduction			
Chapter 2: Background			
Chapter 3: Self-Adaptive and Efficient Configuration of Time Sensitive Networks	Section 3.1	C1	[SEF21]
	Section 3.2	C2	[SEF22]
	Section 3.3	C3	[SF22]
Chapter 4: Resilience against Denial of Service Attacks for Time Sensitive Networks	Section 4.1	C4	[SF23]
	Section 4.2	C5	[SKF23]
	Section 4.3	C6	[SF20]
Chapter 5: Conclusion			

Table 1.1: The organization of the thesis.

schedulability of the time-sensitive flows. We also present and discuss centralized and distributed configuration options.

Furthermore, we have also explored the application of network programmability concepts, such as SDN and P4, to enhance the security of next-generation networks. In [MSEF21], we investigate how programmable network components can be leveraged to detect and prevent router spoofing attacks at the data plane level, thereby safeguarding IPv6 networks. Furthermore, in [SEB+22], we analyze the vulnerabilities in the SDN data plane and propose a trust-based framework to identify compromised nodes.

We have omitted the specific details of those contributions to maintain a coherent storyline focused on the main contributions of this thesis, which primarily pertain to the self-adaptive and efficient configuration and resilient operation of time-sensitive networks.

1.4 Thesis Organization

Throughout the rest of this *cumulative* thesis, we present our publications in the realm of dynamic self-(re)configuration and rersilient operation of time-sensitive networks. Each publication will be presented under the relevant section, aligned with its design goals, and cover the core concepts, key findings, and main takeaways. Since this is a cumulative thesis, we have rephrased and condensed some of the original content. All the original publications can be found in the appendices. The overall organization of the thesis is shown in Table 1.1.

In **Chapter 2**, we will introduce the foundational concepts of time-sensitive networks. We will also provide a brief overview of the TSN protocols relevant to this thesis.

In **Chapter 3**, we present our contributions to the self-adaptive and efficient configuration of TSN. Specifically, Section 3.1 provides an in-depth explanation of the TSN

self-configuration framework (C1). In Section 3.2, we present methods for a dynamic reconfiguration of time-sensitive networks (C2). Lastly, Section 3.3 introduces a ML-based configuration strategy for larger TSN topologies (C3).

Chapter 4 summarizes the contributions to the resilience of time-sensitive networks against DoS attacks. In Section 4.1, we present admission control strategies to maintain strict QoS guarantees in the case of compromised or faulty network entities exist (C4). Section 4.2 summarizes calibrated attacks against TSN frame preemption mechanisms and discusses several countermeasures with respect to C5. In Section 4.3, we present a strategy to differentiate DoS traffic and legitimate traffic; then we present both small and large-scale collaborative attack filtering strategies, as related to C6.

Finally, **Chapter 5** summarizes all contributions and presents further research directions.

1.5 List of Publications

Thesis Contributions

[SEF21] **Sertbaş Bülbül, Nurefşan**, D. Ergenç, and M. Fischer. SDN-based Self-Configuration for Time-Sensitive IoT Networks. *International Conference on Local Computer Networks (LCN)*, 2021.

[SEF22] **Sertbaş Bülbül, Nurefşan**, D. Ergenç, and M. Fischer. Towards SDN-based Dynamic Path Reconfiguration for Time-sensitive Networking. *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022.

[SF20] **Sertbaş Bülbül, Nurefşan** and M. Fischer. SDN/NFV-based DDoS Mitigation via Pushback. *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020.

[SF22] **Sertbaş Bülbül, Nurefşan** and M. Fischer. Reinforcement Learning assisted Routing for Time Sensitive Networks. *2022 IEEE Global Communications Conference (GLOBECOM)*, 2022.

[SF23] **Sertbaş Bülbül, Nurefşan** and M. Fischer. Preemptive DoS attacks on Time Sensitive Networks. *(to appear at) 2023 IEEE Global Communications Conference (GLOBECOM)*, 2023.

[SKF23] **Sertbaş Bülbül, Nurefşan**, J.J. Krüger, and M. Fischer. TSN Gatekeeper: Enforcing stream reservations via P4-based in-network filtering. In *2023 IFIP Networking Conference (IFIP Networking)*, 2023.

Further Contributions

[ESM+23] D. Ergenç, **Sertbaş Bülbül, Nurefşan**, L. Maile, A. Arestova, and M. Fischer. Towards Synthesizing Datasets for IEEE 802.1 Time-sensitive Networking. *2nd International Workshop on Machine Learning and Networking (MaLeNe)*, 2023.

[MSEF21] M. Mönnich, **Sertbaş Bülbül, Nurefşan**, D. 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.

[SEB+22] **Sertbaş Bülbül, Nurefşan**, O. Ermis, Ş Bahtiyar, M. U. Çağlayan, and F. Alagöz. Trust Enhanced Security for Routing in SDN. *2022 1st International Conference on 6G Networking (6GNet)*, 2022.

[SEF22] **Sertbaş Bülbül, Nurefşan**, D. Ergenç, and M. Fischer. Evaluating Dynamic Path Reconfiguration for Time-sensitive Networks. *Würzburg Workshop on Next-Generation Communication Networks (WueWoWas'22)*, 2022.

[SF23] **Sertbaş Bülbül, Nurefşan** and M. Fischer. Adapting to the Flow: Reinforcement Learning for Dynamic Priority Assignment in TSN. *2nd International Workshop on Machine Learning and Networking (MaLeNe)*, 2023.

[WBC+20] F. Wilkens, S. Botzler, J. Curts, S. Dinter, M. Hamann, V. Hubbe, A. Kornivetc, **Sertbaş, Nurefşan**, and M. Fischer. Towards Flexible Security Testing of OT Devices. 2020.

Chapter 2

Background

In this chapter, we provide an overview of IEEE 802.1 time-sensitive networking, including key terminology (Section 2.1) and a concise description of relevant standards (Section 2.2).

2.1 Basics of Time Sensitive Networks

Understanding the fundamental terminology associated with TSN is essential for gaining insight into the key concepts and mechanisms that enable deterministic, low-latency communication over Ethernet networks.

First, the logical flow of time-sensitive data within the network, from one end-host to another, is called *stream*. Each stream has specific QoS requirements and is assigned a unique identifier to differentiate it from other streams. TSN categorizes different types of network traffic into traffic classes based on their priority and QoS requirements such as data delivery guarantee, tolerance to jitter, and packet loss. Thus, each traffic class is treated accordingly during transmission to meet the desired performance goals.

From the architectural perspective, main TSN components are end hosts and bridges. The end host called *talker* is a network device or application that generates and transmits time-sensitive data or streams. Talkers are responsible for initiating the transmission of time-critical information into the network. They typically adhere to specific timing requirements and utilize TSN mechanisms to ensure the timely delivery of their data. The end host called *listener* is also a network device or application that receives and consumes time-sensitive data or streams. Listeners rely on the accurate and timely arrival of data from talkers to perform their intended operations or provide the desired services. Listeners employ TSN mechanisms to synchronize their reception and processing of time-sensitive information.

Lastly, *bridges* in TSN refers to a network device that connects multiple TSN domains or segments. It serves as an intermediary between different time-sensitive networks, facilitating the exchange of time-sensitive data. TSN bridges are equipped with TSN-aware capabilities and employ TSN mechanisms, such as scheduling and prioritization, to ensure seamless and deterministic forwarding of time-sensitive traffic across network segments.

TSN talkers, listeners, and bridges collectively contribute to effectively implementing and operating time-sensitive communication within time-sensitive networks. Talkers generate and transmit time-sensitive data, listeners receive and consume that data, and bridges enable the interconnection and synchronization of time-sensitive traffic across different TSN domains.

In the subsequent section, we provide a concise overview of the classification of some widely adopted TSN standards within the scope of this thesis.

2.2 IEEE Time Sensitive Networking Standards

TSN is not addressed in a single standard; its collection of capabilities is governed and managed by separate IEEE standards and can be classified in four as seen in Figure 2.1. The *time synchronization* standards describe mechanisms to provide a network-wide precision clock reference. The *scheduling* standards specify different mechanisms to limit network delays. At the same time, *policing and redundancy* standards keep non-time-sensitive traffic from messing things up and describe redundancy for the link/node failures. Lastly, the *control and orchestration* standards describe how to configure and maintain these TSN mechanisms.

All standards can be combined as required depending on the requirements of the working environment, e.g., traffic types. While TSN encompasses a broad family of standards,

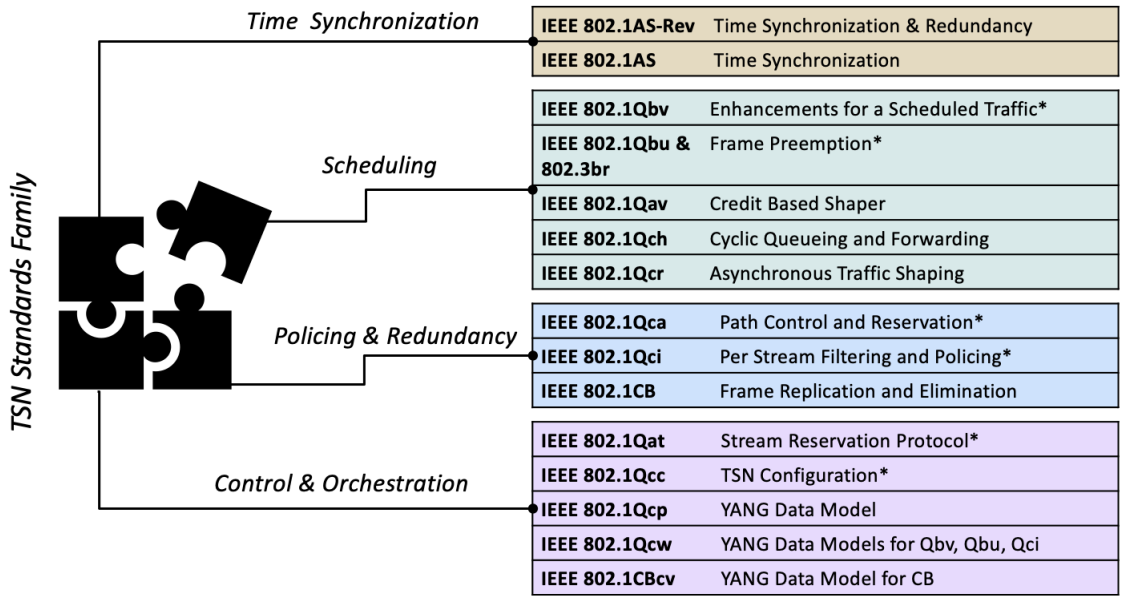
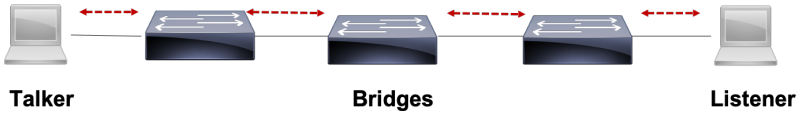
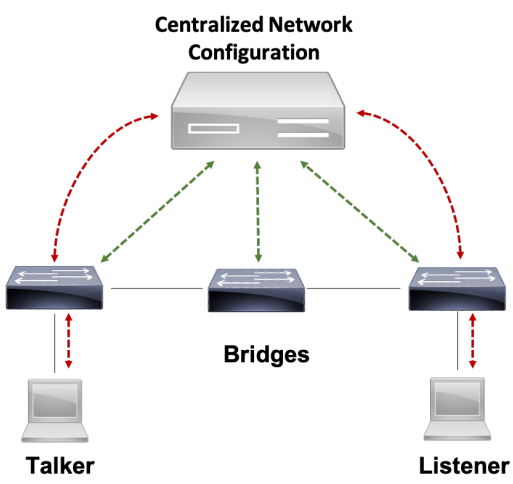


Figure 2.1: IEEE 802.1Q TSN family of standards included in the scope of this thesis (*).

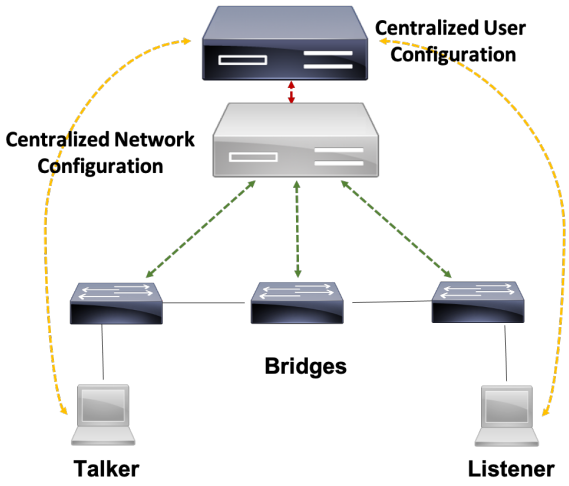
this thesis specifically concentrates on the core standards associated with the configuration and security of time-sensitive networks.



(a) Fully distributed



(b) Centralized network/distributed user



(c) Fully centralized

Figure 2.2: TSN configuration models.

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

The IEEE task group has presented the SRP to describe the resource reservation for specific traffic streams traversing a bridged local area network and thus provide admission control to the time-sensitive network [IEE10]. With SRP, end hosts declare their traffic specification that characterizes the bandwidth a stream can consume to the network. This declaration contains fields such as maximum frame size, stream identifier, max frame interval, and priority. Then, the network reserves the resources to guarantee the transmission and reception of data streams across a network with the requested QoS. Furthermore, SRP also describes other mechanisms for the dynamic maintenance

of those resources in case of new stream registration and de-registration to establish end-to-end stream paths. Therefore, it is one of the essential TSN standards.

To provide specifications for the configuration of TSN features in a SRP, including the SRP configuration, the IEEE TSN task group also presented IEEE 802.1Qcc Enhancements to SRP standard [IEE18]. The propagation of the configuration information within the network depends on the used configuration scheme. The standard proposes three different configuration schemes (i) a fully distributed model, (ii) a centralized network/distributed user model, and (iii) a fully centralized model, as shown respectively in Figure 2.2.

In the *fully distributed* model, as shown in Figure 2.2a, the end hosts (i.e., talkers and listeners) communicate the user requirements directly over the TSN user/network protocol, and these requirements propagate along the topology. The network is configured without a centralized entity; thus, each bridge configures itself with limited local knowledge. In some cases, complete knowledge of the network might be necessary to obtain *best* configuration. Thus, in the *centralized network/distributed user* model, as shown in Figure 2.2b, end hosts still directly talk with the network, e.g., bridge. But the configuration information is directed to/from a centralized network configuration (CNC) entity, and then the resource reservation is made using a remote network management protocol so that the talker/listener information is communicated directly between the edge bridge and the CNC rather than propagating the information to the interior of the network.

Moreover, some TSN use cases, such as industrial control applications, may require detailed knowledge of each end host's application software/hardware. To do so, the *fully centralized* model is as shown in Figure 2.2c, also has a centralized user configuration (CUC) entity that discovers end hosts, retrieves end host capabilities and user requirements, and configures TSN features in end hosts. Here, the user-to-user protocol between the CUC and end hots is left out of the scope of this standard.

IEEE 802.1Qca Path Control and Reservation (PCR)

The path control and reservation (PCR) protocol specifies extensions to the intermediate station to intermediate station (IS-IS) protocol for configuring multiple paths in bridged networks [IEE16b]. It utilizes shortest path bridging (SPB) with a hybrid mode of SDN, where the IS-IS protocol handles fundamental operations, while an SDN controller manages explicit paths through path computation element (PCE)s located on dedicated server nodes. By integrating control protocols such as equal cost tree (ECT) and multiple spanning tree instance (MSTI), IEEE 802.1Qca facilitates the management of multiple topologies, a configuration of explicit forwarding paths for each stream, bandwidth reservation, data protection, redundancy, and the distribution of flow synchronization and control messages. Thus, the IEEE 802.1Qcc can be used in conjunction with the IEEE 802.1Qca and TSN traffic shapers.

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

The IEEE 802.1Qbv time-aware shaper (TAS) is one of the core TSN mechanism that is designed especially to meet the hard deadlines of highly time-critical applications such as industrial control applications [IEE16d]. To ensure high-priority traffic has guaranteed access to a medium at a specific instant, TAS uses a window-based transmission similar to time division multiple access (TDMA) and time-aware transmission gates.

The architectural model of an 802.1Qbv compliant switch is shown in Figure 2.3. Each port in the switch has eight queues, a transmission gate for each queue, and a programmable gate control list (GCL). An incoming frame is processed by the switch and placed into the related queue depending on the traffic class. For instance, the first queue, Q7, is responsible for the mission-critical traffic, while the last queue, Q0, is responsible for the best-effort traffic. A traffic class can access the transmission medium only if the transmission gate of the queue is set to open.

The programmable GCL triggers the opening and closing gate events and operates according to the configured transmission schedule. For the given scenario in the figure, the GCL is configured at the T1 window to allow only gate 7 to be open, and the rest is closed until the next transmission window. Lastly, a transmission selection algorithm decides which frames should be forwarded first, as several streams may exist in the same

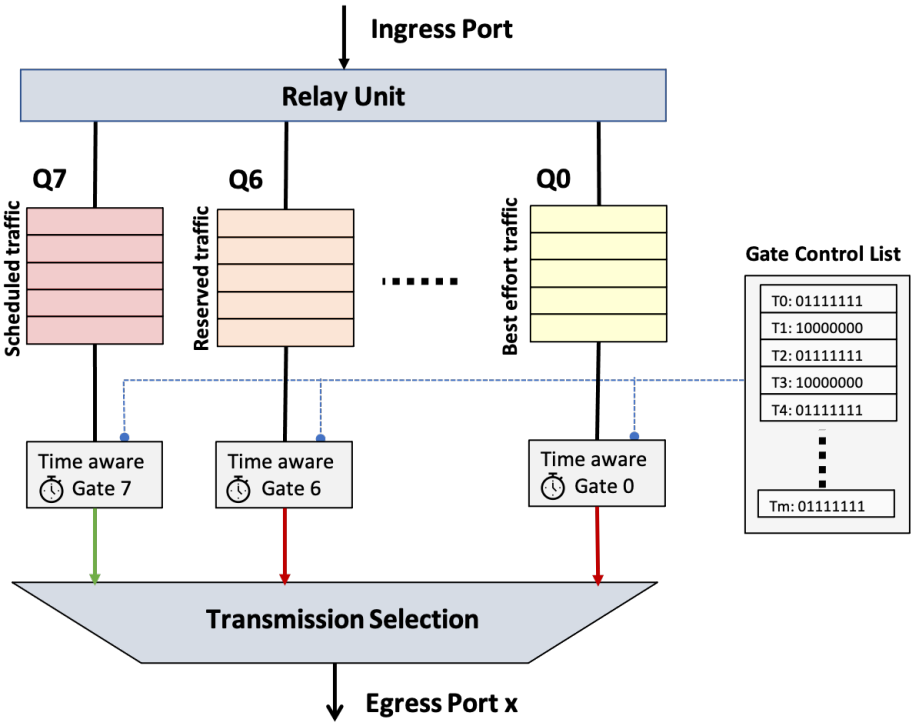


Figure 2.3: IEEE 802.1Qbv time aware shaper mechanism.

traffic class. The time-sensitive network task group does not define a standard method to generate transmission schedules, so it is still an ongoing research area.

IEEE 802.1Qci Per-Stream Filtering and Policing (PSFP)

The IEEE 802.1Qci per stream filtering and policing (PSFP) is the only security standard in the TSN family that enforces some regulations to ensure conformance with the traffic contract of the streams [IEE17b]. It works at the switch's ingress to protect the outgoing queues against faulty and/or malicious endpoints and switches. This kind of stream blocking is essential for TSN as it has to meet and guarantee certain traffic requirements, such as bounded end-to-end latency and zero jitter. For instance, in an autonomous driving scenario where a multitude of sensors need a timely delivery, erroneous input may cause congestion and violation of strict QoS requirements.

Detecting and limiting the faulty stream(s) is necessary to protect the transmission of other non-faulty streams on egress and maintain the promised QoS. This functionality has been conceptually described in the related standard, shown in Figure 2.4, and contains three layers. Initially, the *stream filters* decide which gates and meters handle frames of a specific stream id. Then, depending on the related stream filter configuration, the frame is sent to the related *time-based gate* that ensures timely traffic delivery by filtering based on the configured schedule. Thus, it supports applications where the transmission and reception of frames across the network are coordinated. Finally, as a part of the stream filter configuration, the frame is sent to the related *flow meter*, which enables further fine-grained filtering, such as bandwidth enforcement or frame length-based policing.

Thus, these three layers enable the deployment of different filtering policies and protect against bandwidth violation, malfunctioning, attacks, etc. After the ingress filtering, if that frame is not dropped, it is sent to the shaping and queuing module depending on its traffic priority and finally forwarded to the next hop from the related outport. Here, the standard outlines the overall architecture, while the specific logic employed within these layers remains an open research area.

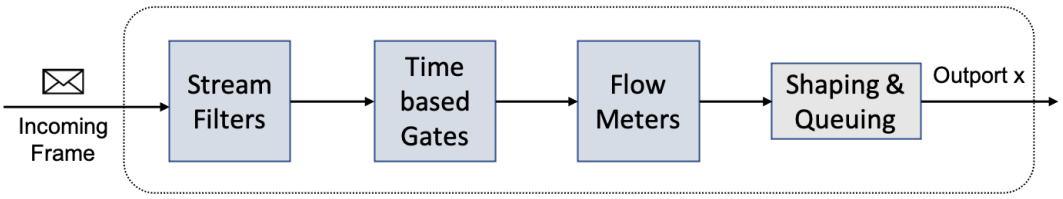


Figure 2.4: TSN switch with the IEEE 802.1Qci ingress filtering.

IEEE 802.3br Interspersing Express Traffic and 802.1Qbu Frame Preemption

The IEEE 802.3br interrupting [IEE16a] and the IEEE 802.1Qbu interspersing [IEE16c] express traffic are TSN mechanisms designed to enhance network performance by alleviating congestion in critical sections of the network. Traffic classes are assigned to either express, non-preemptable, or preemptable. It enables high-priority express traffic to interrupt the transmission of lower-priority preemptable traffic, keeping the shared portion of the interrupted traffic. This allows for a smoother traffic flow on a network and lowers the chance of missing/dropping packets of the critical traffic assigned to an express.

The effect of the frame preemption on the latency of different traffic classes is shown in Figure 2.5. In a standard Ethernet, when high-priority traffic arrives while the low-priority traffic is under the transmission, it has to wait until the current frame transmission is completed, as shown in Figure 2.5a. This may introduce a major delay, as in the worst case, a non-time-critical frame might block a time-critical high-priority frame. However, with the preemption, as shown in Figure 2.5b, high-priority express traffic can interrupt the ongoing transmission of preemptable traffic with a minimal overhead, highlighted with gray, and start transmission. After this transmission, low priority preempted frame can continue to be transmitted. The benefit of this mechanism on the frame transmission latency, the difference between τ_a and τ_b, is considerable, especially for the isochronous traffic class, which requires transmission without interference in a few microseconds.

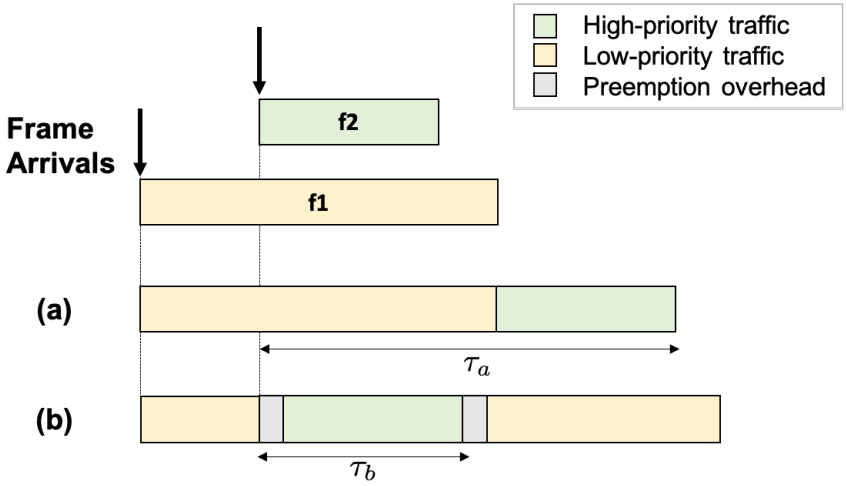


Figure 2.5: Frame preemption effect on the latency: (a) Non-preemptive frame transmission (b) Preemptive frame transmission.

Chapter 3

Self-Adaptive and Efficient Configuration of Time Sensitive Networks

For many years, networks have mostly remained static. Administrators had to plan and configure network components, such as routers, switches, and firewalls. This process was time-consuming and required in-depth expertise. These manual interventions lead to potential downtimes and inefficiencies. In a vision of future networking scenarios, the ability of a network to autonomously adapt its configuration, encompassing parameters and settings, in response to evolving conditions or requirements becomes increasingly significant. Network controllers/administrators may need to deploy specialized network elements to monitor and analyze network traffic to derive performance metrics continuously. Leveraging this information, the network can intelligently fine-tune its configuration to bolster performance, enhance reliability, and seamlessly accommodate the demands of evolving networks.

In the context of TSN, these adjustments can include dynamically allocating network resources, optimizing routing paths based on QoS requirements, and modifying the parameters of network protocols to adapt to changing traffic patterns or the addition/removal of network components. Furthermore, in these networks, strict timing constraints need to be kept.

In this chapter, we summarize our contributions on the *self-adaptive and efficient configuration* of TSN as follows:

- Section 3.1 presents a SDN-based self-configuration framework for TSN that analyzes the network traffic at the edge of the network to automate the resource reservation process [SEF21]. This contribution addresses the autonomous configuration of time-sensitive networks to make TSN transparent by removing end-host-related dependencies and answering `RQ1`.
- Section 3.2 presents different path (re)configuration strategies for improving the resource utilization of time-sensitive networks to avoid sub-optimal flow assignment [SEF22]. This contribution addresses the dynamic configuration of routing paths in TSN at runtime, considering the reconfiguration overhead and answering `RQ2`.

- Section 3.3 presents a RL-based approach that learns the *good* routes to send time-sensitive flows depending on their strict QoS requirements [SF23]. The solutions found by this approach are not necessarily optimal, as finding *best* routes requires solving an optimization problem. This contribution addresses an intelligent configuration of TSN via a ML-driven algorithm and answers `RQ3`.

In the subsequent sections, we outline our contributions regarding the self-adaptive and efficient configuration of TSN, referencing Section 2.2, to bridge the existing gap in the literature. Each section presents (i) a set of fine-grained research questions, (ii) our related publications, and (iii) the summary of contributions and the main takeaways. Furthermore, we emphasize how our work leverages and contributes to relevant TSN standards.

3.1 Dynamic Self-configuration of Time Sensitive Networks

This section introduces a SDN-based self-configuration framework for the TSN to answer how to efficiently make resource reservations while ensuring QoS and reducing end host complexity as formulated in `RQ1`.

Currently used resource reservation procedures in the literature rely on the active participation of end hosts to communicate their traffic requirements before the actual communication. However, for highly heterogeneous environments such as industrial networks, end hosts may not be aware of TSN but still require a certain QoS to be satisfied by the network. Also, the dynamic maintenance of these resources, e.g., flow registrations, requires further signaling and configuration efforts. Thus, configuration mechanisms should ease this process and help to adopt existing TSN protocols for future networking scenarios.

Existing works regarding the configuration of time-sensitive networks either configure the network based on apriori knowledge of the network traffic or consider only specific use cases [NDR16, ASS19]. Thus, we believe that plug-and-play self-configuration might address these research gaps. Accordingly, the following questions arise as a refinement of `RQ1`:

- `RQ1.1`: How to use self-configuration that presumes no active participation of end hosts, and how does it affect the time-sensitive traffic delivery performance?

Moreover, the robustness of the self-configuration framework and its adaptiveness on the runtime is another research direction not addressed in the literature. Hence, the following additional questions arise:

- `RQ1.2`: How accurately can we extract traffic characteristics at the network's edge and of which resource requirements?

These research questions motivated the following publication. In [SEF21], we introduce a self-configuration framework that automates the flow reservation without the need to involve end hosts and compare it with the standard SRP of TSN (See Section 2.2). Our simulation results indicate that 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 flows [1]:

> Nurefşan Sertbaş Bülbül, D. Ergenç, M. Fischer. SDN-based Self-Configuration for Time-Sensitive IoT Networks. IEEE 46th Conference on Local Computer Networks (LCN), 2021.

The remainder of this section presents the details of our SDN-based self-configuration framework and the respective publication [SEF21] is attached in Appendix A.

Methodology

Our self-configuration framework, SC-TSN, eliminates the need for end hosts to declare any traffic requirements before actual communication. This enables more heterogeneous environments, such as industrial networks, where any host, even those not TSN-aware, can join the network.

To provide this functionality, we empower the switches at the edge of a TSN so that they monitor the traffic initiated by the talker and try to extract traffic patterns. During this learning time, all frames belonging to this flow are routed over a default path of the link-weight-based shortest paths. After several observed packets, the edge switch obtains the traffic characteristics of the talker, whether it follows a time-triggered (TT) or best-effort (BE) pattern. The edge switch sends this pattern to the centralized SDN controller, CNC. With the knowledge of the flow's resource requirements, the CNC calculates the optimal path and configures the data plane accordingly. Once this configuration is in place, frames belonging to this flow are routed over the optimized paths, ensuring the flow's desired QoS requirements.

The optimal path computation in our system is formulated as a MILP problem, referred to as time sensitive optimal routing (TSOR). TSOR aims to identify the best end-to-end paths and TAS gate lists for each switch along those paths. It considers the specific requirements of the flow, including resource needs and QoS constraints. For instance, the end-to-end latency should not exceed a certain value, such as the period of the flow. While TSOR is not this thesis's contribution, the focus is primarily on the extraction process of traffic characteristics. For a more comprehensive understanding of TSOR in-

[1] For the given publication, the main contribution belongs to this thesis, including the design of the overall self-configuration framework, its implementation, and evaluation. The second co-author modeled and implemented TSOR with its complexity analysis. The third co-author helped improve the paper's quality with his valuable feedback.

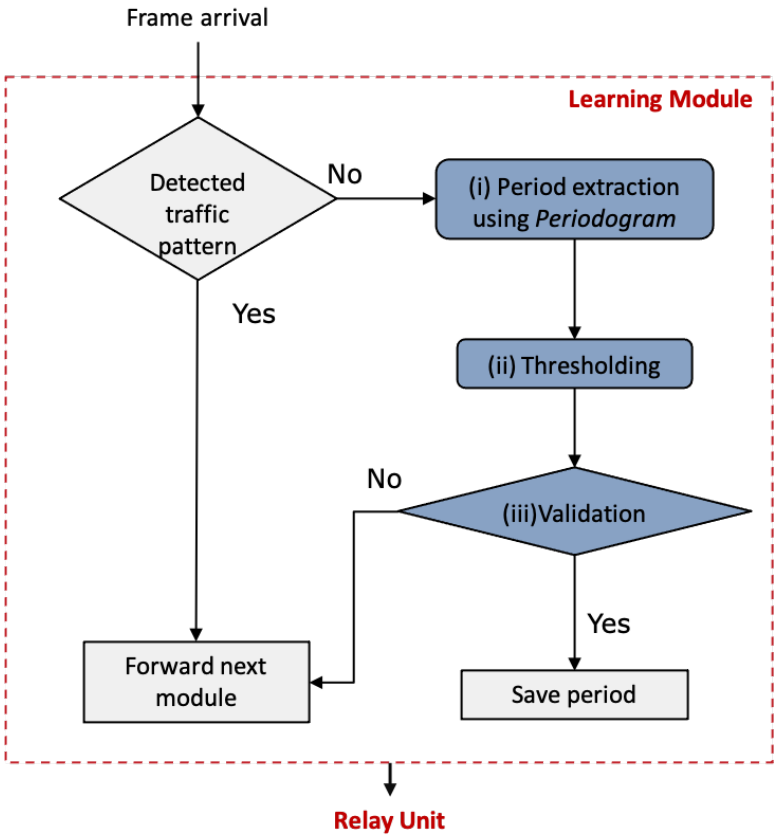


Figure 3.1: Flowchart of the learning module at the edge switches of the time-sensitive network.

cluding its formulation and methodology, please refer to the original publication [SEF21] (Appendix A).

Figure 3.1 shows the flowchart of the learning module we developed for the edge switch. The learning module is positioned at the switch's ingress; once the module processes the frame, it is forwarded to the relay unit, which determines the frame's next hop. In the learning module, the arrival time of frames for each flow is recorded, and the following steps are employed to extract traffic characteristics:

i. It uses a frequency domain periodogram to identify the dominant periods. This involves computing the discrete Fourier transform of the observations to analyze the signal's power spectral density. Then, it can determine the dominant periods by identifying the frequencies that carry the most energy [VYC05].

ii. Relying solely on frequencies may be insufficient due to potential noise interference. To mitigate the impact of noise, it employs the 99% confidence technique to establish a threshold [LDH+10]. Any periods below this threshold are considered misleading and are subsequently removed. This approach helps distinguish

periodicity hints from noise in the Fourier transform.

By following these steps, we effectively extract traffic characteristics while mitigating the impact of noise and false period hints generated by spectral leakage.

iii. In the final step of the period extraction process, it validates the identified periodicity hints using autocorrelation. During this phase, it examines whether the candidate period aligns with the valley of the autocorrelation function. If the candidate period remains within the valley, it is interpreted as a false alarm and subsequently discarded. Conversely, if the candidate period does not align with the valley and exhibits a distinct correlation peak, it is considered a valid period.

Our learning module can further refine and validate the extracted periods by utilizing autocorrelation, ensuring their accuracy and reliability. Then, this period is saved for the flow, and in the remaining frame arrivals, the learning module just checks if the new arrival still obeys the previously extracted traffic pattern.

Key Results

We have comprehensively evaluated our `SC-TSN` framework. For that, we measured the ratio of delayed TT frames that arrived after their deadline. This allows us to assess the effectiveness of the self-configuration mechanism in meeting timing constraints. Furthermore, we measured the accuracy of the learning module in adapting to changing network conditions. We also performed simulations with dynamic traffic scenarios to evaluate the impact of flow bifurcation on the acceptance rate of flows. To provide a baseline for comparison, we also evaluated the native SRP protocol, where talkers announce their resource requirements and a centralized controller employs `TSOR` to configure the TSN switches. By comparing the `SC-TSN` framework with the SRP protocol, we demonstrate that the individual flow's QoS constraints can still be satisfied without the involvement of the end hosts.

We implemented the self-configuration framework using the OMNeT++ network simulator. The simulation model incorporated the self-configuration module. The `TSOR` optimization was implemented separately using CPLEX and then integrated into the simulation. While there is a cost associated with solving optimization problems using `TSOR`, we have chosen to exclude it as we need to solve it for both our framework and the competitor. Furthermore, we assume a fixed processing time for switches to simplify the model, and we believe this to be a reasonable assumption as network elements become more powerful and thus can potentially process data more quickly. More detailed information regarding the simulation parameters and the network topologies employed in our evaluation can be found in the original paper [SEF21](Appendix A).

To create a diverse test scenario, we simulated half of the flows as critical TT and the other half as BE with more relaxed latency requirements. In our experiments, we

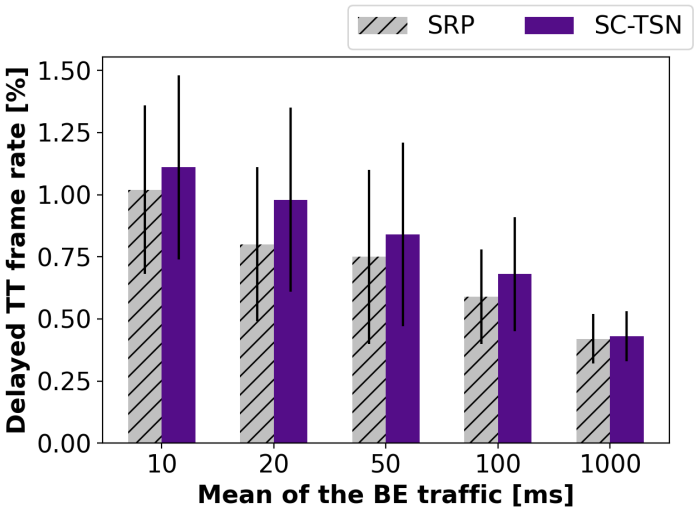


Figure 3.2: Self-configuration impact on the time-sensitive traffic delivery. ©2021 IEEE.

examine the impact of increasing interarrival times of BE frames generated using the Poisson distribution that represents sporadic traffic behavior. Here, BE traffic stands as background traffic, potentially leading to congestion and making scheduling more challenging. Figure 3.2 shows the percentage of the delayed TT frames with a 95% confidence interval. We observe that, across all varying BE loads, the `SC-TSN` framework results in a slightly higher delayed frame ratio compared to SRP, approximately 0.25% higher. In return, we remove a big assumption that end-hosts must initially declare their requirements. It is important to note that this small number of latent frames corresponds to those initially forwarded through the default path, as we anticipated.

For further results regarding the `SC-TSN` framework, the performance of the learning module, and its impact on the time-sensitive traffic delivery, we kindly refer you to the original paper [SEF21] (Appendix A).

👉 Discussion and Implications

Self-configuration of time-sensitive networks is a vital capability that allows them to adapt, optimize, and maintain their performance in dynamic and demanding environments. By automating the configuration process and removing end host involvement, we improve the efficiency of TSN.

As a crucial element of the self-configuration concept, our contribution empowers edge switches in TSN to automatically extract flow characteristics and enable further autonomy in other TSN mechanisms such as TAS. Our experiments show that our proposed self-configuration framework, `SC-TSN`, introduces only a slight additional delay in delivering the first few frames of a flow for making TSN completely transparent to end hosts. The overhead is negligible, especially for long-lasting flows.

However, given the mission-critical nature of some systems, there may be use cases where flows cannot tolerate any additional latency introduced by SC-TSN. In such cases, our self-configuration framework also allows for a SRP-like hybrid configuration, where highly critical flows can be manually configured, while SC-TSN autonomously configures the rest of the flows. With this, we can answer RQ1.

Combining the advantages of automated self-configuration with the flexibility of manual configuration for critical flows, our approach strikes a balance between ensuring reliable communication for time-critical applications and maintaining the efficiency and autonomy of TSN. This approach facilitates the widespread adoption of self-configuring time-sensitive networks, further enhancing their adaptability and performance in real-world scenarios.

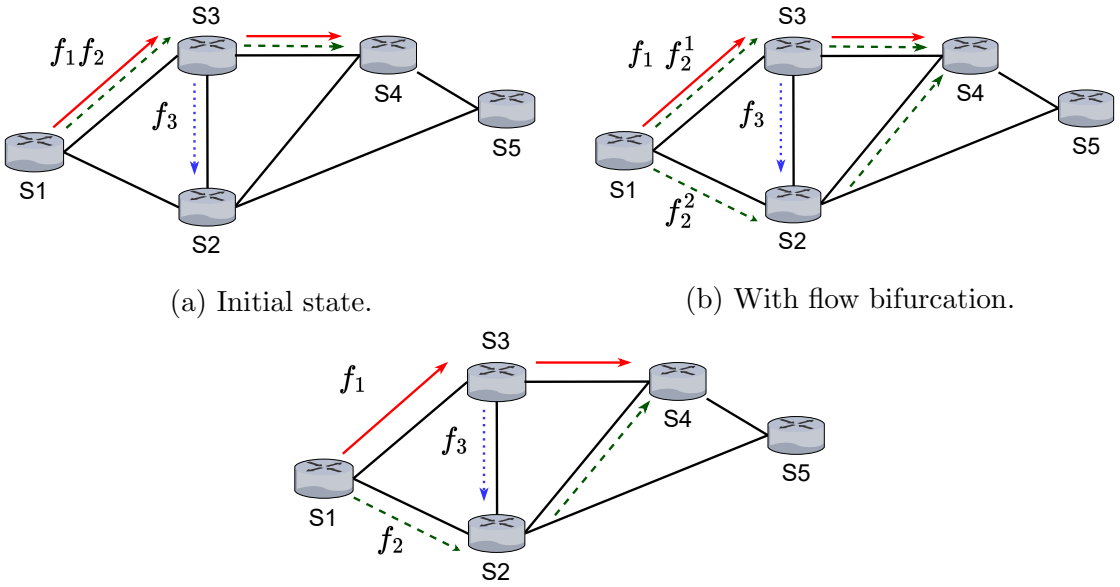
3.2 Reconfiguration Strategies for Time Sensitive Networks

This section introduces different dynamic path (re)configuration strategies for TSN to find out the feasibility of flow migrations in real-time under strict latency constraints, as we also formulated in RQ2.

The dynamic traffic scenarios in which traffic characteristics such as period may change during the network's lifetime imply a change in the required network resources. This change can manifest in two ways: a decrease or increase in the resource requirements. If the demand's resource requirement decreases, the CNC can adjust the resource reservations accordingly without requiring any modifications to the routing. This flexibility allows for efficient resource utilization. If the demand's resource requirement increases, e.g., more bandwidth due to shorter sending periods, the CNC can update the resource reservations accordingly. However, in such a scenario, the current placement of flows within the network may not permit a straightforward increase in resources for the repetitive flows. This situation may require a re-evaluation of the flow placement and potentially an update on the routes.

We present a simple scenario in Figure 3.3 to provide a better illustration. In this scenario, there are three flows: f_1 (represented by the red straight line), f_2 (represented by the green dashed line) and f_3 (represented by the blue dotted line). For simplicity, let's assume that all flows have a bandwidth requirement of one unit and all links in the network have an identical capacity of two units. Initially, in Figure 3.3a, all flows can be easily accommodated in the network as the total required bandwidth does not exceed the capacity of the links. However, let us consider a situation where the bandwidth requirement of f_2 increases from one unit to two units due to dynamic traffic behavior. Due to the limited link bandwidths, f_2 can no longer be routed over the existing path. To address this, there are two potential approaches:

Flow bifurcation: One option is to bifurcate the flow, as shown in Figure 3.3b. In this case, the flow is divided (as f_2^1 and f_2^2) and sent over two different routes, each of which



(a) Initial state.

(b) With flow bifurcation.

(c) With flow migration.

Figure 3.3: Dynamic flow handling scenarios.

can still satisfy the desired QoS constraints. This approach allows for the utilization of multiple paths to handle the increased bandwidth requirement, and its feasibility within the context of the TSN environment is explored in [ZWY+21].

Flow migration: Another possibility is to migrate f_2 to another path as represented in Figure 3.3c, which we discuss further details in the remainder of this section. By finding an alternate path to accommodate the increased bandwidth requirement, f_2 can continue to be routed without violating the desired QoS constraints.

Flow migration enables load balancing by redistributing network traffic to alleviate congestion and enhance overall performance. In case of network failures or link congestion, it helps maintain connectivity by redirecting flows along alternative paths. A resource utilization can deteriorate over time as flows are placed as they arrive. When flows are not migrated, this results in sub-optimal flow assignment. Thus, (re)configuration of *some* of the network resources like link bandwidth *time to time* would be a reasonable solution.

Even though flow placement considering QoS is one of the most important design problems in TSN, existing literature has limited coverage regarding the incremental addition of flows. Moreover, many relevant papers assume that routing paths are pre-determined and do not address the aspect of a path (re)configuration [NDR17, YCW22]. Therefore, the following questions arise in the context of `RQ2`:

- `RQ2.1`: What will be the best strategy to (re)configure the network that minimizes (re)configuration cost while maximizing the network utilization?

- RQ2.2: How do reconfiguration strategies impact the timing requirements of the traffic?

The research questions form the basis for examining performance aspects of the dynamic path (re)configuration in TSN, specifically focusing on delivering time-sensitive traffic and network utilization. Accordingly, the following publication introduces different (re)configuration approaches and compares their performance, scalability, and as well as their potential real-world applications [2]:

> Nurefşan Sertbaş Bülbül, D. Ergenç, M. Fischer. Towards SDN-based Dynamic Path Reconfiguration for Time-sensitive Networking. IEEE/IFIP Network Operations and Management Symposium, 2022.

In the rest of this section, we present the details of proposed path (re)configuration strategies. The respective publication [SEF22] is also attached in Appendix B.

Methodology

Managing the resources in TSN is challenging, given the strict QoS constraints and dynamic traffic nature. To ensure an efficient utilization of resources, it is necessary to adjust the network. As discussed in Section 3.1, flow migration can be used for resource optimization. It allows for the dynamic relocation of flows within the TSN based on the current network conditions, such as congestion levels or changes in network topology. By migrating flows, the network can adapt to varying demands and optimize the allocation of resources with respect to QoS requirements.

By leveraging flow migration, time-sensitive networks can achieve several benefits. First, it enables efficient resource utilization by dynamically reallocating flows to less congested or more suitable paths. This reduces the risk of bottlenecks and improves overall network performance. Second, flow migration enhances the network's ability to react to changing traffic patterns or the network topology, e.g., as a result of node or link failures, ensuring adaptability and flexibility. Finally, it contributes to meeting the QoS requirements of time-sensitive applications by optimizing resource allocation in real-time. However, it is important to note that flow migration also introduces challenges, such as disruptions and additional latency during the transition of a flow to a new path.

Therefore, to test the feasibility of real-time flow migration in TSN, we propose three path (re)configuration strategies. To set a baseline for comparison, we use the original

[2]For the given publication, the main contribution belongs to this thesis, including the design, implementation, and evaluation of the reconfiguration framework including the proposed heuristics. The same time-sensitive optimal routing model, TSOR, which the second co-author modeled, has been re-used in this study in the form of TSOR-U and TSOR-R. The third co-author helped to improve the paper's quality with his valuable feedback.

optimization problem, TSOR, which is already introduced in Section 3.1, and then present alternative strategies.

Restricted path reconfiguration, TSOR-R: The strict time constraints of time-sensitive environments necessitate the allocation of flows on specific paths that meet the required QoS constraints. Once a path satisfies these requirements, it remains unchanged to ensure consistent and reliable communication. This pre-assignment constraint is an integral part of the optimization problem introduced in Section 3.1, which we call TSOR-R, representing the time-sensitive optimization problem with restricted configuration. Including this constraint in our optimization problem serves the purpose of maintaining a stable configuration scheme, particularly for critical and high-priority demands. While this constraint may reduce the flexibility of routing options, ensuring the predictable and timely delivery of time-sensitive traffic is crucial. By keeping the previous assignments fixed, TSOR-R guarantees a consistent configuration that minimizes disruptions and fluctuations in the network. This stability is of utmost importance in time-sensitive environments, in which even slight variations in latency can have significant consequences.

Reconfiguration at every path request, TSOR-U: One strategy to maximize the number of flow embeddings using TSOR is replanning all path configurations from scratch. This involves removing the pre-assignment constraint from TSOR, resulting in an unrestricted version called TSOR-U. This way, all flows in the network can be reconfigured, including their migration to different paths and changing the gate configurations to achieve optimal allocation, considering newly arriving flows. However, this unrestricted approach comes with additional costs, including the delay of signaling to the controller, computation of solutions, and possible packet loss during the migration. This additional configuration time may introduce variability and potential delays in the network, affecting the timely delivery of time-sensitive data.

Reconfiguration at every k-th path request, TSOR-P: One approach to address this additional path computation time is to utilize TSOR-R for embedding new flows in the network. However, continuously using TSOR-R for each flow request may lead to suboptimal resource utilization, particularly when dealing with many flows. To overcome this issue, we propose an alternative solution called TSOR-P (time-sensitive optimization problem with periodic reconfiguration). TSOR-P involves periodically reconfiguring the network after a certain number of flow requests, denoted as by k.

Reconfiguring the assignments periodically helps to adapt to changes in traffic patterns and optimize the allocation of network resources. Instead of reconfiguring after every flow request, TSOR-P waits until a sufficient number of requests have accumulated. This strategy reduces the frequency of reconfiguration while ensuring timely adaptations to the changing flow demands. Furthermore, the reconfiguration period in TSOR-P can

be enhanced by monitoring the network statistics and analyzing the number of late packets. The imbalanced resource utilizations due to join/leave operations of flows lead to increased late packets. In such scenarios, it may be a sensible approach to reduce the value of k to increase the frequency of network reconfigurations.

Threshold-triggered reconfiguration, `TSOR-T`: A straightforward strategy that a network operator can employ is to compute both `TSOR-R` and `TSOR-U` to measure how close the resulting solutions for flow assignments are. If they differ by more than a threshold, τ an unrestricted reconfiguration along `TSOR-U` is carried out. This strategy called `TSOR-T` (time-sensitive optimization problem with threshold-based reconfiguration), allows for reconfigurations based on a pre-defined threshold.

This threshold-based approach in `TSOR-T` provides a practical mechanism for network operators to determine when a reconfiguration is necessary. It provides a trade-off between `TSOR-U` and `TSOR-R`, as it regularly evaluates the current configuration's proximity to the optimal allocation.

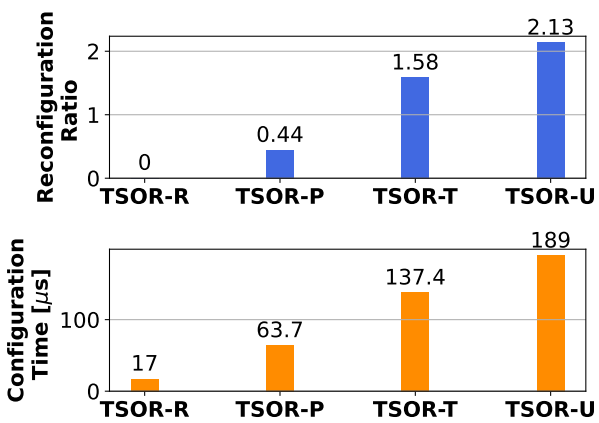
Key Results

We evaluate our reconfiguration strategies by assessing the flow acceptance ratio and the resulting reconfiguration overhead in a realistic simulation via OMNeT++. The acceptance ratio refers to the proportion of flows that can be successfully embedded. A flow cannot be embedded if no end-to-end path satisfies a particular flow's QoS requirements. Consequently, the acceptance ratio serves as a measure of resource utilization efficiency.

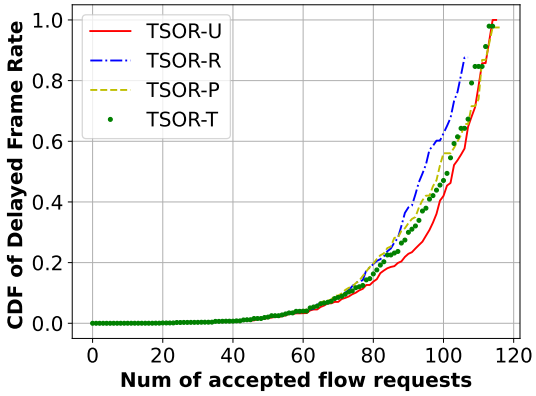
To evaluate the different strategies, we utilize the same simulation setup as described in Section 3.1 and more details in our original publication [SEF22] (Appendix B). Our simulations employ a traffic mix of TT and BE flows and use different distributions to pick their data rates. This approach allows us to demonstrate the dynamic increase in communication demand within a resource-constrained system. Our primary objective in these experiments is to investigate the trade-off between the acceptance ratio and the reconfiguration overhead while measuring the time-sensitive delivery performance of the proposed heuristics: packet delay and loss.

Figure 3.4a (top) plots the reconfiguration overhead for the different strategies. The reconfiguration ratio indicates the proportion of reconfigured flows to the total number of flows. The configuration time, as depicted in Figure 3.4a (bottom), represents the time elapsed until the reconfiguration has been completed.

`TSOR-R` does not reconfigure the existing flows and thus has a reconfiguration ratio of zero. `TSOR-U` has the highest reconfiguration ratio since it reconfigures for each arrival of a new flow for reconfiguration. The results of `TSOR-T` and `TSOR-P` are in between others. The parameter selection influences their effectiveness. For the decreasing the threshold



(a) Reconfiguration overhead for TSOR



(b) Missing deadline ratio with respect to number of accepted flows

Figure 3.4: Performance evaluation of flow migration in time-sensitive network. ©2022 IEEE.

in TSOR-T and k values in TSOR-P would approximate the results to TSOR-U. As the configuration time is closely linked to the number of reconfigurations, the findings align with the reconfiguration ratio.

Furthermore, we also evaluate the time-sensitive traffic delivery performance of these strategies, as depicted in Figure 3.4b. In this evaluation, we considered a frame delayed if its end-to-end latency exceeded the deadline of the flow. When comparing the strategies with the same number of accepted flows, we observe that TSOR-U exhibits the lowest delayed frame rate, which supports our claim. However, TSOR-R had the highest delayed frame rate, as it lacked the ability to re-evaluate the current resource distribution, resulting in limited opportunities to select optimal paths for new flows. Both TSOR-P and TSOR-T outperform TSOR-R and closely approach the performance of TSOR-U. However, it is important to note that beyond a certain threshold, around 110 flows in this particular experiment, the number of delays significantly increased in TSOR-U. This phenomenon occurs when certain links get highly utilized, leaving limited capacity for accommodating new flows.

More detailed analysis of different scenarios can be found in the original publication [SEF22](Appendix B).

Discussion and Implications

The presented TSOR strategies seem to enable a balance between reconfiguration overhead and seem to provide good QoS at low reconfiguration overhead for various use case scenarios. Our evaluation highlights the significance of considering the trade-offs between resource optimization and flow acceptance when choosing the most suitable TSOR strategy. This understanding is crucial for aligning the strategy with the specific require-

Strategy	Reconfiguration Trigger	Flexibility	QoS	Configuration Overhead
TSOR-U	After every flow	High	Medium	High
TSOR-R	No reconfiguration	Low	High	Low
TSOR-P	After every kth flow	Medium	Low	Medium
TSOR-T	Exceeding predefined threshold	Medium	Low	Medium

Table 3.1: The summary of path (re)configuration strategies. ©2022 IEEE.

ments of a given network environment. Network administrators can effectively manage resources by considering these trade-offs and ensure optimal QoS for time-sensitive traffic.

Accordingly, in Table 3.1, we summarize the proposed heuristics regarding their flexibility in resource use, resulting QoS, and reconfiguration overhead. A proper reconfiguration strategy can be selected depending on the requirements of the environment. For example, reconfiguring at every path request would be an option for utilizing all resources in a highly dynamic small or medium-scale environment where flows are added and removed over time. However, its computation overhead and cost of deployment make this approach impractical. It would be only meaningful to do this if the current placement is too far away from the optimal placement. Alternatively, reconfiguring at every kth path request and reconfiguring only when the solution deviates more than a threshold from the optimal solution seems to result in 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 promising reconfiguration solutions for time-sensitive scenarios. In assessing the feasibility of the real-time reconfiguration considering the reconfiguration overhead, we can answer RQ2.

3.3 Machine Learning-based Intelligent Configuration of Time Sensitive Networks

This section introduces a RL-based routing method that determines the routing path for diverse scenarios and addresses RQ3. It adaptively evaluates paths and considers the flow deadline constraints to ensure effective routing decisions.

Configuring routing in TSN to maintain QoS requirements becomes challenging with many constraints, such as scheduling-driven delays and strict flow deadlines. Traditional shortest-path routing algorithms are inadequate as they result in high latency as a result of congestion and, thus, violation of flow deadlines. Their slow convergence speed is also unsuitable for dynamic networks [WC92]. In contrast, researchers have utilized centralized network control paradigms such as SDN integrated with optimization.

However, in particular problems such as joint routing and scheduling, this becomes NP-hard and exhibits high computational complexity, particularly for large networks [FDR18].

Here, RL-based routing could be an alternative solution to the inabilities of traditional routing and the complexity issues in ILP-based routing approaches. This gives rise to the following research question in addition to `RQ3`:

- `RQ3.1`: Can RL be used for flow allocations in TSN? To what extent can it approximate the optimal solution, and what are the associated trade-offs?

This research question is the foundation for utilizing reinforcement learning to accommodate flows in TSN considering the QoS. For the remainder of this section, we present the details of our RL-based routing approach that we name as `RL` and its design choices.

Accordingly, the following publication introduces RL assisted routing approach specifically designed for TSN and compares its performance with other benchmarking algorithms such as shortest path and integer linear programming[3]:

> Nurefşan Sertbaş Bülbül and M. Fischer. Reinforcement Learning assisted Routing for Time Sensitive Networks. IEEE Global Communications Conference (GLOBECOM), 2022.

The respective publication [SF22] is also attached in Appendix C.

Methodology

Our reinforcement learning-based routing solution, `RL`, can provide near-optimal solutions for the time-sensitive routing problem while significantly reducing computational complexity compared to ILP. However, one notable shortcoming of reinforcement learning is its often extended convergence time, which we do not have when solving the optimization models. Despite the potential for extended convergence times, reinforcement learning offers the advantage of adaptability and the ability to learn complex decision-making processes, leading to robust and flexible solutions once the learning phase is completed.

`RL` uses Q-learning, a popular reinforcement learning algorithm based on trial and error learning. In Q-learning, an agent interacts with states and takes actions to maximize cumulative rewards. The Q-learning algorithm provides a foundation for training intelligent agents to make sequential decisions in dynamic environments. It employs the so-called Q-table to store the estimated value of each state-action pair. The agent updates the Q-table through repeated iterations by applying the Bellman equation, which

[3]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.

calculates the expected future rewards. By iteratively exploring and exploiting the environment, the agent gradually learns the optimal policy to maximize long-term rewards. This procedure is illustrated in Figure 3.5 visually.

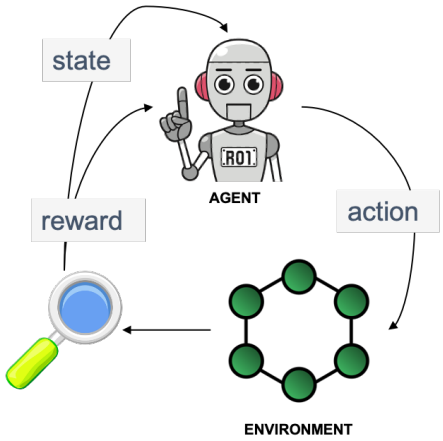


Figure 3.5: Simplified RL model where an agent interacts with the environment to maximize long-term rewards through state transitions.

Reinforcement-learning-based time-sensitive routing To deploy `RL`, we use a centralized SDN controller by leveraging its global view to employ centralized routing algorithms and simplify configuration processes. This framework allows us to dynamically reconfigure routing paths with respect to the requirements of time-sensitive flows.

Our SDN-based framework is shown in Figure 3.6. After the talker requests resources for a flow, the request is forwarded to the controller, and the `RL agent` determines the routing path based on its current policy. This policy is dynamically updated based on the feedback from the data plane (e.g., TSN switches) that is included as a *reward*. The decision of the `RL agent`, the so-called *action*, refers to the routing path in our design, for which the controller then configures the data plane, e.g., TSN switches accordingly. In this design, a *state* is a traffic matrix representing the current network load, and with each *action*, the system transits from one state to another. Then, the talker can start to communicate over the reserved path. In the background, the last switch on the path reports statistics, such as end-to-end latency of the frame, to the controller as feedback. Based on the statistics, a *reward* is calculated for the associated action and routing path, which is used to update the `RL` model.

We aim to acquire the optimal routing policy that maximizes the cumulative reward over time. To achieve this, a crucial step is defining a reward function that guides the policy towards the desired behavior: finding the good routes to satisfy individual flow deadlines in our specific environment.

To prioritize deadline satisfaction rather than latency, we have established a unique reward function received upon successful frame reception. The reward, R is calculated

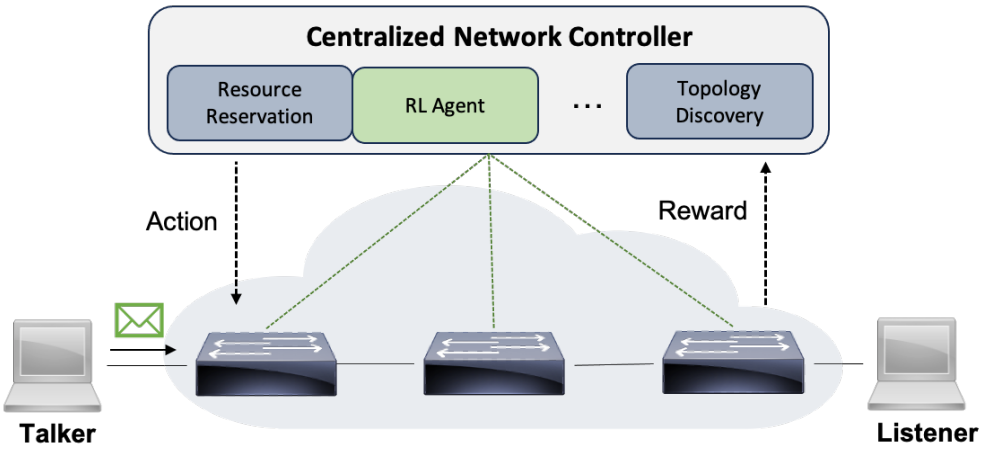


Figure 3.6: RL: reinforcement learning based routing approach for TSN

as follows:

$$R = \begin{cases} 1 + \frac{\Delta t}{D}, & \text{if } \Delta t \geq 0 \\ \frac{\Delta t}{D}, & \text{otherwise} \end{cases} \tag{3.1}$$

where D represents the flow's deadline, which we assume is equivalent to its period. Δt is the remaining time until the deadline, which can be computed by taking the difference between the end-to-end latency, $t_{E2Elatency}$, of the frame and its deadline. Here, the reward values for the actions (e.g., selected path) can be either positive or negative. We define the reward for the non-negative Δt values by dividing Δt by the deadline D so that flows with closer deadlines get a greater reward for the same Δt values. Additionally, to give a positive reward in case the deadline is met, even though Δt is zero, we add 1. For the negative Δt values that represent the missed deadlines, we use Δt proportional to D as a penalty, which is also a negative number. After enough iterations, the reinforcement learning algorithm integrates the negative penalty values into the learned policy to guide the agent away from undesirable actions. Therefore, we can choose paths for flows that offer a higher deadline satisfaction rate for newly registered flows. This tailored approach allows Q-learning to effectively learn the optimal routing policy while considering the important aspect of meeting flow deadlines.

Algorithm 1 shows the pseudo-code of our algorithm that runs on the SDN controller. The aim is to develop a policy to select a path from origin *src* to destination *dst*. Initially, when there is a path request, the ϵ-greedy policy chooses one action from the vector of feasible actions (Line 3-8). The algorithm explores the feasible action space with a probability ϵ. Also, since there is no pre-knowledge about the paths, it searches for a path with a higher reward with a probability of $1 - \epsilon$.

Algorithm 1: Q-learning based routing - control plane

Input: Learning rate: α
Discount factor: γ
Exploration and exploitation parameter: ϵ
List of paths: $\mathcal{P}$
Network link-states: $\mathcal{S}$

1: Initialize Q: Q(s,a)
2: **while** Path request received from *src* to *dst* with a deadline D **do**
3: Generate a random number m
4: **if** $m < \epsilon$ **then**
5: select random action, $a = \mathcal{P}[rand(0, numOfPaths)]$
6: **else**
7: select action, $a = Q_{max}(s, a)$
8: **end if**
9: Install forwarding rules for the selected action a (aka path)
10: **end while**
11: **while** Reward update message from the data plane **do**
12: Updates Q-table and moves a new network state:
13: $Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma max_{a_{t+1}Q(s_{t+1}, a_{t+1})}]$
14: **end while**
15: **return** Optimal routing policy

The ϵ value is usually chosen small so that the policy may take advantage of knowledge about the present state most of the time. After the action selection, the SDN controller enforces related forwarding rules to the data plane, and the end host starts the transmission (Line 9).

When frames over this path are received by the destination switch S_{dst}, it computes the reward for this transmission as shown in Algorithm 2 and sends it to the controller. The SDN controller uses the reward to update the Q-table for the related path with the equation given in Line 13 in Algorithm 1.

The given Q-table formulation represents how an agent updates its estimate of the expected cumulative reward $Q(s_t, a_t)$ based on the immediate reward, $R(s_t, a_t)$, that is

Algorithm 2: Q-learning based routing - data plane (e.g., switch S_{dst})

Input: Received frame, f, from *src* to *dst* with a deadline D

1: **while** f.destination == S_{dst} **do**
2: $t_{E2Elatency} = f.arrivalTime - f.creationTime$
3: $\Delta t = D - t_{E2Elatency}$
4: $R = \begin{cases} 1 + \frac{\Delta t}{D}, & \text{if } \Delta t \geq 0 \\ \frac{\Delta t}{D}, & \text{otherwise} \end{cases}$
5: Computed reward, R, is sent to the controller
6: **end while**

received after taking an action (a_t). Here, the formulation also includes the maximum expected cumulative reward in the next state (s_{t+1}) and the learning rate (α) and discount factor (γ) parameters. The learning rate parameter determines how much new information is incorporated into the existing Q-value. Smaller α values make learning more stable but slower, while larger values make learning faster but less stable. The discount factor parameter represents the agent's preference for short-term rewards over long-term rewards. Smaller γ values make the agent focus more on immediate rewards, while larger values focus more on long-term rewards.

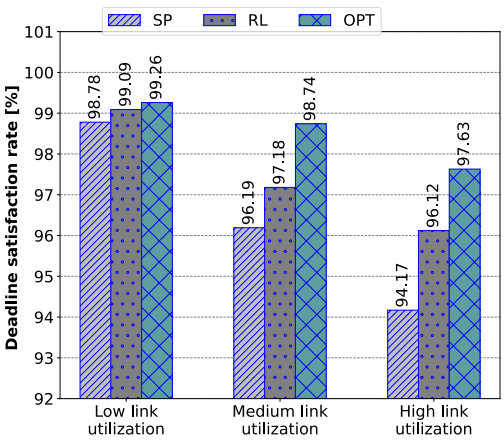
Key Results

We comprehensively evaluate our reinforcement learning-based framework, referred as `RL`, via OMNeT++ simulations and in realistic TSN scenarios. To assess the performance of `RL`, we compare it against two benchmarking solutions: shortest path (`SP`) and optimal routing via integer linear programming (`OPT`).

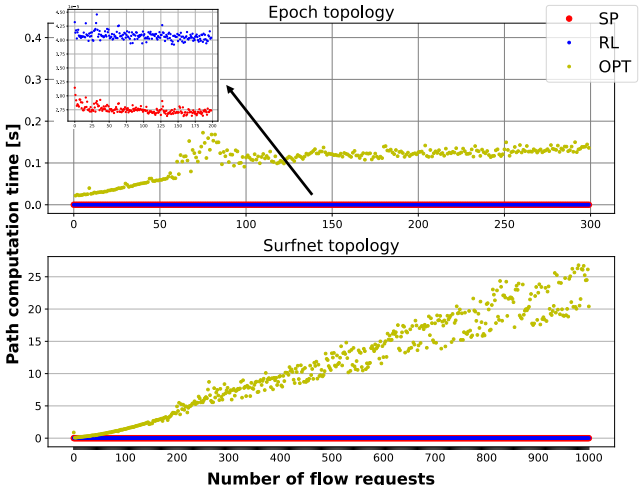
The `SP` approach directs traffic along the shortest paths, often leading to congestion on specific switch ports or links within the network. Due to its simplicity and minimal computational requirements, we utilized `SP` as a baseline to compare it against our proposed `RL` approach. Conversely, `OPT`, which we introduced in Section 3.1, as an optimization problem `TSOR`, determines the optimal allocation of resources and routing paths. We consider `OPT` as an upper bound for evaluating the effectiveness of `RL`, measuring how closely `RL` can approach the optimum solution.

To evaluate our approach, we create various mixed traffic scenarios that include both TT and BE traffic types, each with different QoS requirements. Additionally, we test our framework on topologies of different sizes to assess its scalability. For more in-depth information regarding simulation parameters and specific details, please refer to our corresponding publication [SF22] (Appendix C). Our evaluation aims to assess the performance and effectiveness of our `RL` framework within the context of TSN environments.

To evaluate the delivery performance of our `RL` model and benchmarking approaches, we utilize a medium-scale topology. Initially, we embed best-effort flows to saturate the network resources. Subsequently, we generate 180 TT flows for a low average link utilization (approximately 15%), 450 TT flows for a medium average link utilization (approximately 45%), and 900 TT flows for a high average link utilization (approximately 75%). The deadline satisfaction rate of the TT traffic was then measured, as shown in Figure 3.7a for the different link utilizations. As expected, congestion significantly decreases the satisfaction rate in the high utilization case (e.g., from 99.26% to 97.63% in `OPT`). Notably, the satisfaction rate in SP experiences a more substantial decrease, dropping from 98.78% to 94.17% due to its limited ability to use alternative paths during congestion. However, the RL algorithm is close to OPT, with a 96.12% deadline satisfaction rate. In other words, only 3.88% of the frames fail to meet their deadlines, even in higher utilization scenarios.



(a) Time-sensitive traffic delivery performance.



(b) (Incremental) path computation time.

Figure 3.7: Performance comparison of RL with other benchmarking approaches. ©2022 IEEE.

Moreover, we conduct experiments using small-scale (e.g., Epoch) and large-scale (e.g., Surfnet) topologies to assess the scalability of the algorithms. For that, we measure the path computation time while progressively increasing the number of flow requests. The results, presented in Figure 3.7b with a 95% confidence interval, demonstrate interesting insights. Notably, both SP and RL approaches, based on table look-up, exhibit similar path computation times, even for large topologies like Surfnet. However, the path computation time of the OPT solution remains acceptable for small topologies, as can be seen in the top Figure 3.7b, but significantly increases as the topology grows larger (See Figure 3.7b-bottom). These results support our claim that OPT may not be suitable for large-scale scenarios due to its runtime complexity. Here, it should also be stated that RL has an implicit cost, which is the convergence time referring to the number of iterations needed for RL agents to learn an effective policy that achieves a satisfactory level of performance.

For additional results concerning other QoS metrics, including the deadline satisfaction rate of individual traffic classes, as well as end-to-end latency and jitter, we refer readers to the original publication for more comprehensive details [SF22] (Appendix C).

👉 Discussion and Implications

Our evaluation provides several interesting highlights. First, using reinforcement learning for efficient path allocation in flows with individual QoS constraints proves to be a promising solution. By relying on the environment's feedback, it avoids the need for a priori knowledge about the network, making it suitable for real-world scenarios. The achieved deadline satisfaction rate of 96.12% indicates the effectiveness of the proposed

approach, even in the highly utilized cases. This rate closely approaches the optimum and outperforms the congestion-prone shortest path routing method.

Moreover, the computation time of the RL remains constant as the network size increases, in contrast to the exponentially growing complexity of the MILP solution. This highlights the scalability of the RL, making it a viable option for larger networks where MILP becomes impractical.

The proposed approach aligns well with the evolving concept of self-configuration in networking. It offers flexibility and adaptability in diverse network conditions, showing promise for future network deployments and dynamic topology changes. However, further research is needed to enhance the resilience of the RL algorithm under varying network conditions and to consider the transmission scheduling aspect of TSN. Moreover, employing helper techniques, e.g., experience replay, where past experiences are stored and re-included in the model, or using different exploration strategies to balance exploration and exploitation may fasten the convergence time of the algorithm, which is often crucial in real-world scenarios.

Overall, the findings of this study demonstrate the potential of RL in addressing the efficient path allocation problem in networks with individual QoS constraints and answers RQ3.

Chapter 4

Resilience Against Denial of Service Attacks for Time Sensitive Networks

Time-sensitive networks must maintain continuous and reliable data transmission without significant delays or disruptions. This has been made possible by deploying various fine-tuned time-based mechanisms standardized by the IEEE 802 TSN task group, which ensures that certain QoS. One unique attack vector for TSN environment is the time itself. Attackers can cause violation of strict QoS requirements by introducing significant delays. That might not directly impact availability but can negatively affect the resilient operation of critical applications. Moreover, these attacks cause congestion and undermine the network's ability to provide predictable and low-latency communication. Therefore, in order to maintain such guarantees, it is necessary to avoid the operation of any faulty or malicious end hosts or switches.

Despite its significance in mission-critical systems like automotive, avionics, and industrial networks, resilience has not been considered sufficiently in the design of TSN. While severe cases can arise due to this oversight, addressing potential threats and implementing appropriate countermeasures is essential. Hence, this chapter describes our contributions towards the *resilient operation* of time-sensitive networks. For that, this thesis makes the following contributions:

- In Section 4.1, dynamic admission control strategies based on P4 are presented as a link-layer network function [SKF23]. This solution defends time-sensitive networks against malicious network elements and faulty talkers or switches. This contribution addresses the enforcement of all end-hosts to obey their reserved resource limits considering the filtering overhead and answers the `RQ4`.
- In Section 4.2, DoS attacks against the TSN frame preemption mechanism and their potential implications are summarized. The section describes an attack strategy to degrade the QoS of TSN and shortly discusses potential countermeasures [SF23]. This contribution shows the feasibility of attacks even with passive network monitoring answers the `RQ5`.
- In Section 4.3, a SDN/NFV based reactive attack filtering [SF20] is presented [SF20]. It first efficiently describes the attack signature by distinguishing the attack and

legitimate traffic and then filters the attack traffic. This contribution helps to protect the network from flooding network elements and answers the RQ6.

In the subsequent sections, we present our contributions within this domain, referencing Section 2.2, to bridge the existing gap in the literature. 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. We will emphasize how our work leverages and contributes to the relevant TSN standard.

4.1 Admission Control Strategies for Time Sensitive Networks

This section introduces different admission control strategies for filtering malicious or faulty traffic at the ingress of the TSN switches and addresses RQ4. These strategies aim to satisfy desired QoS by preventing the network from unintended traffic.

TSN requires end hosts to pre-register their traffic requirements, ensuring the allocation of essential resources along the entire end-to-end path. This proactive approach involves prior agreement between the end host and the network before communication occurs. Thus, the network can guarantee a specific QoS for the requesting end host, talker. All elements within the network must adhere to their designated resource limits, e.g., using only the allocated bandwidth. Malicious talkers that exceed their reserved traffic volume can induce congestion of switches along the path. This congestion can violate the bandwidth guarantees of legitimate flows. Thus, the ingress traffic should be checked and verified at the ingress of the TSN switch.

The IEEE task group has proposed the IEEE 802.1 Qci standard as a solution for ingress traffic filtering that surpasses its reserved resources. However, the current state of the standard remains largely conceptual, lacking a concrete algorithm for implementation [IEE17b]. Only a limited number of studies have proposed algorithms compatible with the standard. However, these studies often assume that the configuration parameters are set initially and remain unchanged throughout the network's lifespan [LWF+21]. Alternatively, they may focus combination solely on specific TSN mechanisms [MHKS19]. Such use-case-driven or static configurations limit the potential benefits of the protocol's dynamic capabilities. Thus, this presents an ideal use-case scenario for implementing dynamic and adaptable filtering solutions using P4, a versatile and programmable networking language. Since P4 provides fine-grained control over packet processing and enables dynamic modifications, it is a promising tool for implementing customizable security functionalities. Therefore, the following more fine-grained questions arise in the context of RQ4:

- RQ4.1: How does the proposed P4-based filtering approaches perform compared to no filtering applied regarding end-to-end latency and frame loss?

Describing the traffic that needs to be filtered is a crucial step in the filtering procedure. However, in the context of TSN, ingress filtering may not necessitate the extraction of these attack signatures, as it relies on resource reservation in advance.

Accordingly, the following publication introduces P4-based ingress attack filtering framework to avoid the malicious attempts that might affect the delivery of the legitimate time-sensitive traffic [1]:

> Nurefşan Sertbaş Bülbül, J.J. Krüger, M. Fischer. TSN Gatekeeper: Enforcing stream reservations via P4-based in-network filtering. The International Federation for Information Processing (IFIP) Networking Conference, 2023.

In the remainder of this section, we present the details of our local admission control strategies. Respective publication [SKF23] is also attached in Appendix D.

Methodology

P4-based In-network Attack Filtering As a main working principle of TSN, resources are reserved before the actual communication to guarantee stringent QoS. It is assumed that every entity in a time-sensitive network obeys these reserved limits. However, malicious or faulty talkers might exceed their reserved bandwidth. Our P4-based ingress filtering framework, `TSN Gatekeeper`, restricts all talkers to their reserved bandwidth and maintains QoS for the resilient operation of the network.

Figure 4.1 gives an overview of `TSN Gatekeeper`, which filters the unauthorized traffic at the network's ingress. For that, `TSN Gatekeeper` implements two P4-based filtering strategies aligned with the TSN's IEEE 802.1 Qci standard. These strategies differ in handling traffic, e.g., per-class or per-flow (aka per stream). Here, deciding how to filter is another design choice, and `TSN Gatekeeper` proposes two policing strategies: thresholding or blocking. When filtering according to a threshold, traffic up to the advertised limit is forwarded, and any exceeding traffic is blocked. This has clear benefits in certain situations, such as a temporarily faulty end host can be kept in the network while effectively containing its threats. An alternative is to block the flow entirely when it exceeds the advertised limit. Unlike the first strategy, all traffic is dropped at the ingress, even when the flow later returns to its initial parameters.

Before giving the details of the individual strategies, it is important to mention that P4-based security offers several advantages. It provides flexibility due to programmable network devices, enabling the implementation of security protocols tailored on TSN

[1] For the given publication, the main contribution belongs to this thesis, including the design of the system in the context of his master's thesis. The second co-author has implemented and evaluated the designed system. The third co-author helped to improve the quality of the paper with his valuable feedback

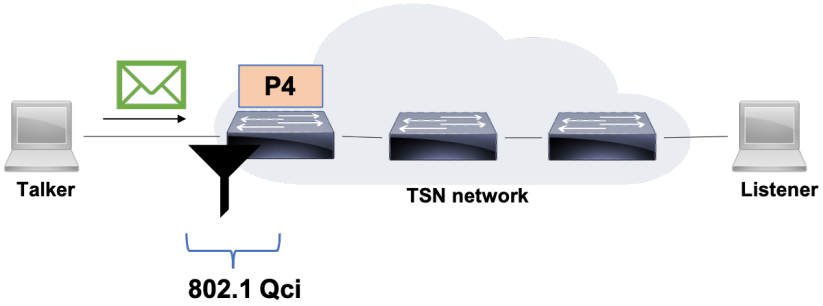
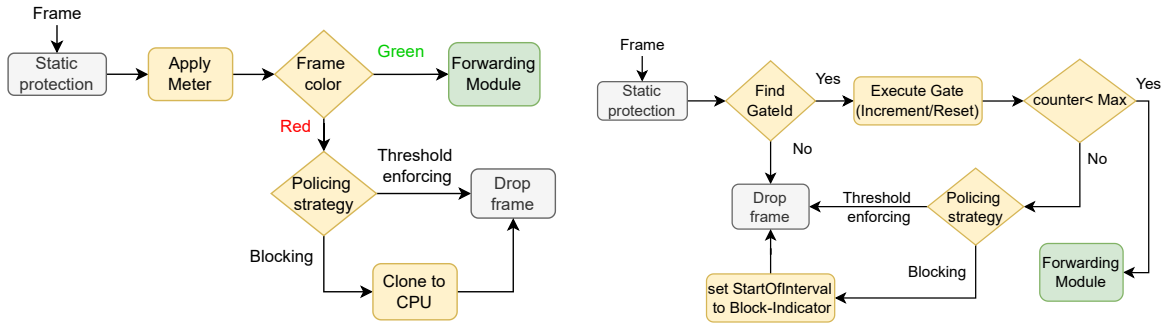


Figure 4.1: Simple time-sensitive network where switches are empowered by the P4-based filtering capability to enforce flow reservations in the network's ingress.

requirements. Furthermore, it enables quick updates and modifications to security protocols in response to changing network conditions and evolving threats. This enables the precise handling of network traffic, including admission control and filtering.

As a first step, `TSN Gatekeeper` initially checks the maximum frame size and the ingress port of the received frame, which is already negotiated as a part of the resource reservation process (See SRP in Section 2.2). Frames that exceed the maximum size are dropped to prevent potential switch congestion. To prevent attackers from flooding frames with spoofed StreamIds, e.g., to disrupt or interfere with the transmission of legitimate talkers, ingress port verification is conducted. So that the frames that come at the wrong ports are dropped. Following these initial verifications, `TSN Gatekeeper` deploys two dynamic filtering solutions, namely metered ingress filtering and gated ingress filtering, which are described as follows:

(a) Metered ingress filtering: The deployment of metered ingress filtering benefits from the portable switch architecture (PSA), a target architecture that provides standard data types, counters, meters, and other externs for P4 programmers. The PSA ensures the portability of P4 programs across different targets. We use direct meters from the PSA primitive and propose ingress filtering based on the leaky bucket algorithm, where the burst size, *BS*, defines the initial bucket size. The bucket size is then increased by the pre-configured information rate, *IR*, per second and decreased upon a packet arrival. The *IR* parameter can be computed using the talker-advertise message, representing the number of frames per measurement interval. The *BS* can be interpreted as the maximum number of frames by which a flow can exceed the advertised rate. This parameter proves useful in handling frame delays that may require the transmission of frames, even for legitimate flows. When the *BS* falls below zero, packets are marked as red; otherwise, they are considered green. We attach a direct meter to the forwarding table to implement this filtering behavior. It is automatically executed when a matching entry exists. The burst size and information rate parameters can be configured per table entry through the P4Runtime API, which serves as the



(a) Flowchart of the metered ingress filtering (b) Flowchart of the gated ingress filtering

Figure 4.2: Flowcharts of the proposed filtering approaches.

conceptual control plane. It is important to note that this control plane belongs to P4 and is not a centralized controller.

A flowchart is given in Figure 4.2a. After the initial checks, the metered-based filtering marks the frame as either red or green. The green frames will be forwarded accordingly, while red frames are dropped based on the policing configuration, either thresholding or blocking. While the frame is directly dropped in thresholding, CPU cloning informs the control plane in the blocking. This way, the controller can block the flow and do a de-registration.

(b) Gated ingress filtering: P4 also supports registers as a general-purpose data type that facilitates the implementation of fully customized algorithms. Unlike meters, registers cannot be used per-table-entry or per-flow (aka per-stream) basis. To address this limitation, we use a concept named *gates*, inspired by the per-class filtering approach in the IEEE 802.1Qci standard. A gate merges the traffic characteristics of multiple flows and treats them as a single flow. By employing more gates, we achieve finer-grained filtering capabilities, albeit with increased memory requirements.

It is important to note that gates do not enforce per-flow policies but work based on the per-class filtering principle. As long as the total traffic at a gate does not exceed its capacity, the transmission of any individual flow is not restricted. In other words, the allowed transmission capacity for a gate may not be distributed equally among the flows assigned to that gate. However, this approach is reasonable as it aligns with the class-based queuing delays in TSN. Thus, gate selection becomes an important design principle here. For instance, deterministic gate selection algorithms can be exploited by attackers to specifically target certain flows by injecting traffic to those gates. Therefore, using a hash function or similar deterministic gate selection methods is unsuitable. Thus, for the sake of simplicity, we follow the fill empty first (FEF) approach that fills empty gates first so that any flow violation will have a limited effect on others as it is also limited to a

particular gate. Such an approach may suffer if many flows exist on the gate. However, the design of a proper gate algorithm is left out of scope.

A flowchart is given in Figure 4.2b for gated ingress filtering. After the initial checks, the frame is handled by the gate determined by the gate selection algorithm. Then, a frame counter, which increases with each incoming frame, is checked to determine whether it exceeds the *Max* value, which is given by the controller. To accommodate the varying intervals and the number of frames sent by different traffic classes in TSN, we adopted a common observation interval and adjusted the frame count accordingly. This counter within the given observation interval operates as a bandwidth check for the gate, as we know the frame size from the flow reservation. With that, it rejects and drops frames that exceed the predetermined bandwidth. Thus, the frame is forwarded if the current counter fits that limit. Otherwise, depending on the policing strategy, it is dropped. However, unlike metered ingress filtering, this blocking mechanism operates without involving the control plane, allowing gates to be closed at the line rate. Additionally, this approach requires less memory since it does not require copying packets to the CPU for further control plane processing.

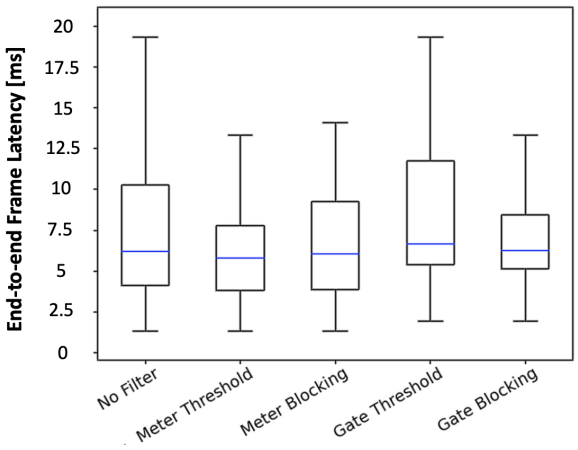
For further details and explanation, please refer to our original publication [SKF23] (Appendix D).

Key Results

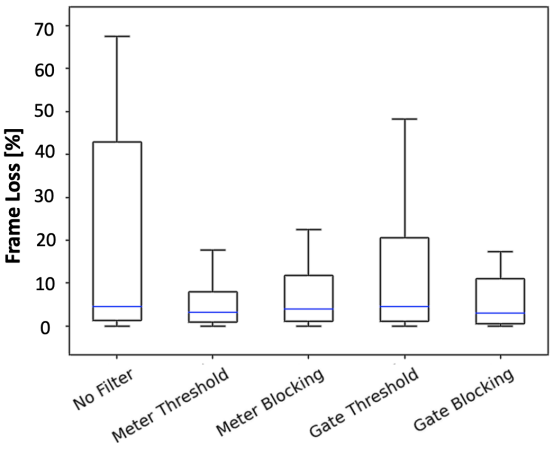
The ingress filtering approaches within `TSN Gatekeeper` are implemented using the P4 behavioral model version 2 (bmv2), emulating attacks on Mininet. It is worth noting that the software switch bmv2 used may not be specifically designed for performance evaluations and may not accurately reflect the performance of the mechanisms on a hardware switch. However, for a fair comparison between the approaches, it is suitable.

As attacker model, we use a babbling idiot representing a talker who correctly advertises traffic and receives a corresponding listener-ready message but then sends more traffic than advertised and thus exceeds the allocated bandwidth. We use a four-switch ring topology and placed end hosts randomly in the network. Different traffic types are generated for realistic experiments, including isochronous, cyclic, event-triggered, and best-effort traffic, which are typical TSN traffic classes. In our experiments, we evaluate the performance of four different filtering strategies implemented within the `TSN Gatekeeper`. These strategies include metered and gated approaches configured with either thresholding or blocking. We compare the performance of these approaches with the scenario where no filtering was applied. Results of the filtering performance in terms of frame loss rate and end-to-end frame latency are shown in Figure 4.3.

Figure 4.3a shows end-to-end frame latencies dependent on different filtering approaches as box plots. It can be observed that the average latencies among the tested approaches



(a) End-to-end latency for mixed traffic scenarios



(b) Frame loss for mixed traffic scenarios

Figure 4.3: Filtering performance on the delivery of time-sensitive traffic.

do not exhibit significant differences. However, meter-based filtering with thresholding or blocking results in lower and more consistent end-to-end latencies, which is significant for TSN. It is, however, important to note that solely considering latency values might be insufficient, as it only reflects the end-to-end latency of successfully transmitted frames.

Hence, examining the frame loss metric is also important and even of greater significance for TSN. In Figure 4.3b, a significant number of frame losses can be observed due to the babbling idiot. Merely considering average values would be misleading since TSN guarantees a certain QoS, necessitating the consideration of worst-case frame loss rates. Without filtering, the attacker impacts legitimate flows substantially, leading to frame losses of up to 68% as in the *No-Filter* case. However, it is important to note that all filtering approaches significantly reduce the frame loss rate compared to this maximum value. In the worst-case scenario, the filtering approaches decrease the frame loss rate for legitimate flows to approximately 24% (in the upper bound of *meter-blocking*) and approximately 48% (in the upper bound of *gate-thresholding*). This demonstrates the substantial benefits of these approaches.

For further results regarding the impact of an increasing number of attackers, refer to our original publication [SKF23] (Appendix D).

👉 Discussion and Implications

Our P4-based in-network attack filtering framework, `TSN Gatekeeper`, is tested in an emulated mininet environment. The results indicate that `TSN Gatekeeper` effectively reduces frame loss rates for legitimate traffic while adding only minimal filtering overhead. Consequently, it has the potential to meet the stringent performance requirements in time-sensitive environments.

Furthermore, the proposed filtering approaches align with the IEEE 802.1Qci standard. The flexibility, programmability, and low-latency capabilities of P4 make it an excellent choice for attack filtering, enabling precise and adaptable security measures to be applied to network traffic. This design also eliminates the need for a centralized controller, which is crucial for handling attacks in real-time. Moreover, thanks to P4, `TSN Gatekeeper` can be easily extended to support further filtering criteria, such as filtering based on burst sizes or maximum frame lengths.

4.2 Calibrated Attacks Against TSN Frame Preemption and Countermeasures

This section analyzes the feasibility of the calibrated attacks against TSN frame preemption mechanism by describing a few attack scenarios. To answer `RQ5`, it demonstrates how an attacker can leverage insights gained from passive monitoring to degrade the network performance and discusses potential countermeasures.

Frame preemption temporarily halts the transmission of lower-priority frames when a high-priority frame arrives. This capability ensures the timely delivery of time-sensitive traffic, such as real-time control messages or audio/video flows, and minimizes disruptions caused by lower-priority traffic. Furthermore, it facilitates the coexistence of different traffic types, supporting time-sensitive and non-time-sensitive applications on the same network infrastructure.

Determinism within TSN can pose a security threat as attackers may exploit extended periods of network monitoring to gather valuable insights. For instance, they can monitor the network and keep track of frame transmissions, observing when a critical frame is being forwarded by a specific TSN node. This may result in unexpected long delays and violations of QoS constraints. Therefore, analyzing the current frame preemption protocol against calibrated attacks is crucial to enhance the resilient operation of the network and maintain desired QoS levels.

However, the preemption mechanism is considered one of the core mechanisms in time-sensitive networking for bounding latency and has not been viewed as a security vulnerability [AHG21, HFG+20]. So far, there is a lack of discussion on the vulnerabilities of the preemption mechanism from the attacker's perspective in the existing literature. Therefore, the following questions arise in addition to the `RQ5`:

- `RQ5.1`: How can an attacker monitor the network to estimate the used preemption scheme in the switch?
- `RQ5.2`: How can passive monitoring information be leveraged to degrade the QoS for time-sensitive traffic, and how it impacts the end-to-end latency of the critical traffic?

Accordingly, the following publication introduces a comprehensive analysis of frame preemption protocol against calibrated attacks such as DoS attacks and discusses several preventive measures to mitigate these attacks [2]:

> Nurefşan Sertbaş Bülbül and M. Fischer. Preemptive DoS attacks on Time Sensitive Networks. Submitted to IEEE Global Communications Conference (GLOBECOM), 2023.

In the remainder of this section, we present the details of an attack on frame preemption on time-sensitive traffic. The respective publication [SF23] is attached in Appendix E.

Methodology

Throughout the study, we describe various attack scenarios based on different configurations of traffic classes, e.g., preemptable or express. Then, we discuss their potential impacts on network performance. Following this analysis, it is essential to assess the potential information that an attacker could extract from the network and to explore how it can utilized to attack the network. Understanding the impact of a successful attack on critical traffic delivery is crucial for designing countermeasures to mitigate such attacks effectively.

Frame Preemption Scenarios Since preemption categorizes the traffic as express or preemptable, four possible scenarios may occur. We describe these scenarios to analyze the possible configurations and examine their potential implications thoroughly. By that, we can gain insights into the behavior and consequences of preemption within the TSN framework. The four scenarios, illustrated in Figure 4.4, are as follows:

(a) Express frames block preemptable frames: The frame preemption mechanism can significantly decrease waiting times of high-priority traffic. However, it is important to note that frames can be preempted multiple times, resulting in additional overhead due to the frame-splitting process. In such scenarios, switches require enhanced processing capabilities to identify and handle preemptable frames. Additionally, within the frame payload, adding control information, as illustrated as a gray block in Figure 4.4, is necessary to facilitate the resumption of preempted frame transmissions. This overhead can considerably delay the frame transmission and potentially lead to the starvation of preemptable frames and the depletion of switch buffers. As demonstrated in Figure 4.4a, even though the preemptable frame f_1 arrives before the express frames, f_3, f_4, f_5, it has to wait. As in this example, sending consecutive express frames results in multiple preemptions of f_1 and long delays. Also, the other preemptable frame, f_2, faces a similar long

[2] For the given publication, the whole contribution belongs to this thesis. The co-author helped to improve the quality of the paper with his valuable feedback.

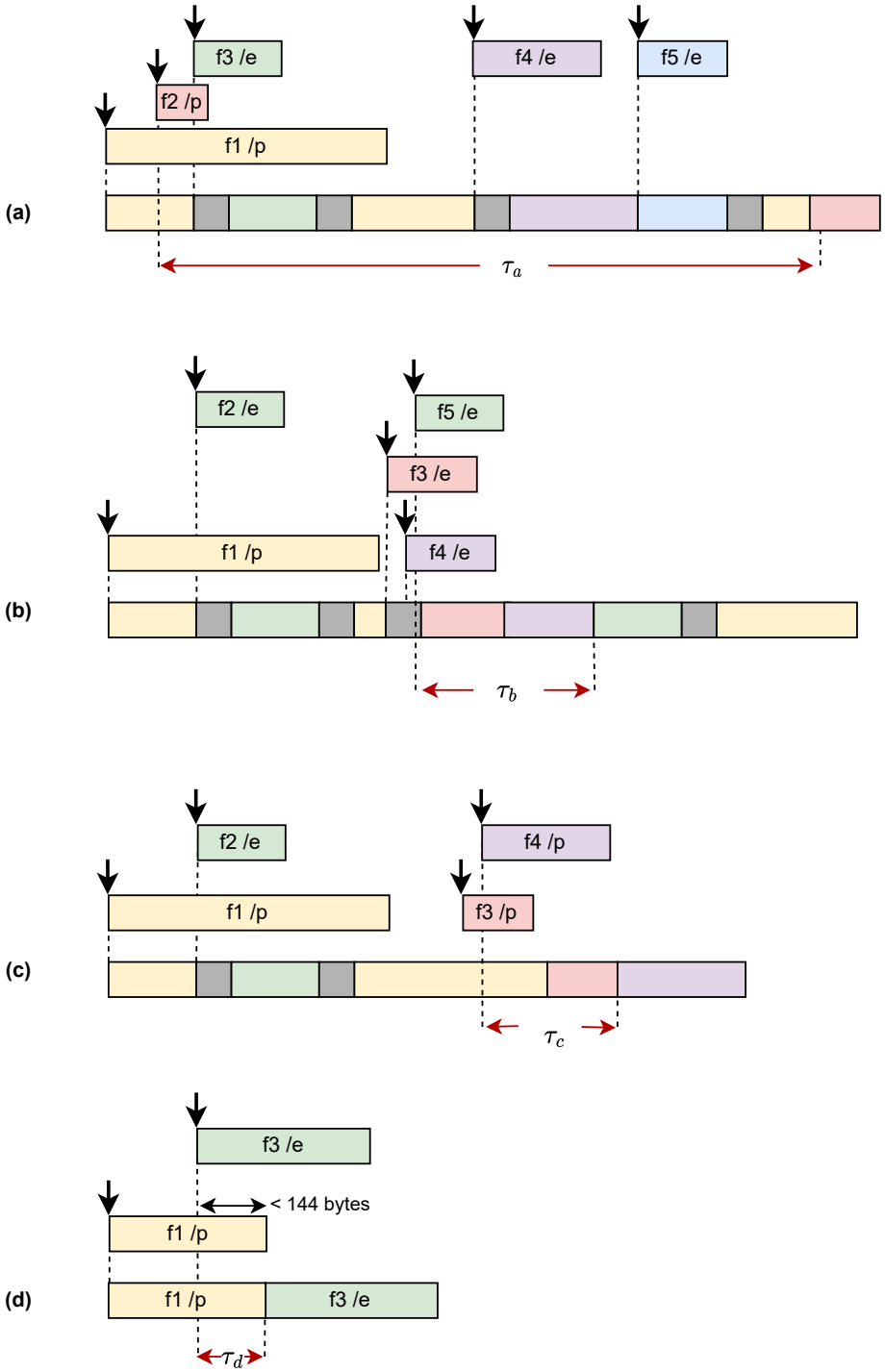


Figure 4.4: Preemption effect on particular scenarios: (a) express frames to block preemptable frames, (b) express frames delaying each other, (c) preemptable frames delaying each other, (d) preemptable frames to block express frames. ©2023 IEEE.

waiting time. This highlights the challenges associated with multiple preemptions

and a potential impact on traffic delivery.

(b) Express frames delay each other: Although express frames cannot preempt each other, they can still introduce delays that impact time-sensitive traffic. In Figure 4.4b, an attacker can exploit this by sending specific priority frames before receiving a high-priority time-critical frame. Let us consider the scenario where the green frames occur periodically, and the attacker has extracted this period. By strategically inserting higher priority frames before resuming transmission of f_2 (specifically, before f_5), the attacker can cause f_5 to wait for the transmissions of f_3 and f_4. This additional delay imposed by the attacker, adjusted based on the extracted transmission period, can induce a violation of the timing constraints of f_5. In the worst-case scenario, all express flows in the network utilize the same egress port, increasing the potential impact of these delays on time-sensitive traffic.

(c) Preemptable frames delay each other: According to the standard preemption mechanism, preemptable frames cannot preempt each other. When these preemptable frames are forwarded based on a first-come, first-served approach, as recommended by the standards, some frames may experience prolonged waiting times. Although these frames are classified as preemptable, they may still possess relatively softer timing constraints [OYN20]. However, attackers can exploit this situation by sending a long preemptable frame before transmitting lower-priority frames. As depicted in Figure 4.4c, if another preemptable frame, denoted as f_3, is inserted just before the transmission of a time-critical, yet low-priority, preemptable frame, referred to as f_4, the latter frame will be forced to wait until both f_1 and f_3 have been transmitted. Consequently, the attackers' deliberate insertion of frames can significantly delay the delivery of lower-priority preemptable frames, undermining their timely transmission.

(d) Preemptable frames block express frames: While express/preemptable frames cannot preempt express frames, this rule has certain exceptions. According to the preemption standard, further preemption is not permitted if the fragment size fails to meet the minimum Ethernet frame size requirement. Consequently, in the worst-case scenario illustrated in Figure 4.4d, the express frame must wait to transmit 143 bytes, the longest non-preemptable size. This restriction implies that the express frame experiences delay due to the minimum frame size requirement despite the preemption mechanism in place.

Black Box Time-Sensitive Network Traffic Analysis The described scenarios show the feasibility of executing a calibrated attack when the network traffic and the configuration scheme (preemptable or express) are known. This requires that the attacker has complete knowledge of the network. Accordingly, we investigate the worst-case attack that attacker can observe the ingress and egress of the switch, as the TSN nodes are black boxes for the attacker. By analyzing the captured traffic, the attacker can extract the

time differences between transmitted messages (e.g., frames) and other traffic parameters such as priority and period. Additionally, the attacker may employ a traffic-analysis attack to ascertain when a particular node forwards a specific frame or determines its processing location at a given time.

Based on passive network observation, the attacker can see the transmitted frames, their traffic priority, and their ingress and egress time to/from the TSN switches. With this, the attacker can calculate minimum, maximum, and average latency as well as the variance values per traffic class. However, considering the different traffic scenarios, the variance of the traffic class serves as a reliable metric that helps the attacker to estimate the employed frame preemption configuration. For example, Figure 4.5 represents an observation result where only the priority seven traffic is configured as express traffic, while the others are preemptable. Thus, only the priority seven traffic has a constant variance independent from the incoming traffic class. As illustrated in the figure, other preemptable classes experience changes in variance values regarding the injection of different traffic classes. Therefore, here, the determinism of TSN increases the predictability of the network. Then, the attacker may perform traffic injection at a particular time to a certain TSN node with a desired traffic class to carry out a calibrated attack and degrade the QoS of the network. This particular time can be selected depending on the target frame transmission, e.g., just before the transmission of the preemptable frame to preempt it.

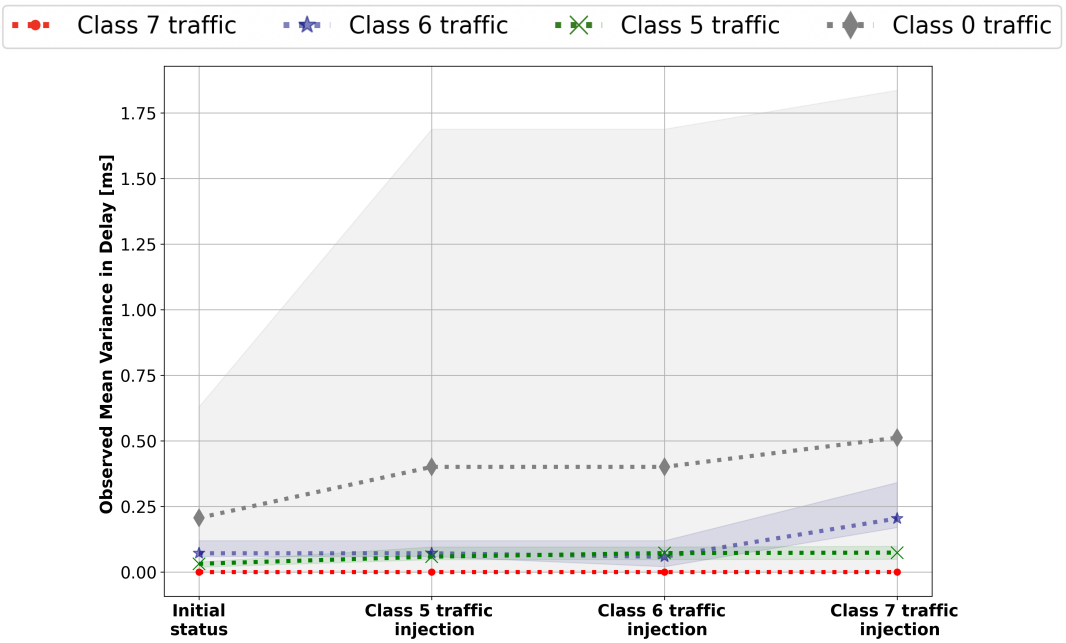


Figure 4.5: Attacker observations regarding only class seven traffic is configured to express. ©2023 IEEE.

Key Results

We have evaluated frame preemption via extensive simulations in in-vehicle network use cases. For that, we have employed OMNeT++ as our simulation tool. We use four distinct traffic classes: control signals, sensor data, raw data from cameras or radars (lidar), and best-effort traffic. In this context, the best-effort traffic does not require specific timing guarantees and represents non-time-critical traffic. The remaining traffic classes contain time-critical traffic. Therefore, these data flows have predefined deadlines. Further details about the simulation environment and the parameters can be found in the original publication [SF23] (Appendix E).

With the help of passive network monitoring, the attacker obtains the used configuration scheme in frame preemption and extracts traffic characteristics of the talker, as shown under the self-configuration concept in Section 3.1. An active attacker can inject malicious traffic into the network based on this information.

The illustrative scenario shown in Figure 4.4a describes the attacker injecting express frames as bursts to block preemptable frames. Accordingly, this scenario has been tested in a realistic TSN environment, and the results are shown in Figure 4.6. The results indicate that even a single express flow, which generates multiple bursts periodically, can cause a significant delay of preemptable traffic, S1 and S2. In the worst case, preempting several times may cause the starvation of the preemptable traffic. This is not visible in the figure but can be inferred from our simulation results.

Moreover, Figure 4.7 shows that an attacker can add further delays to express traffic by injecting express traffic as in the scenario shown in Figure 4.4b. Specifically, the initial

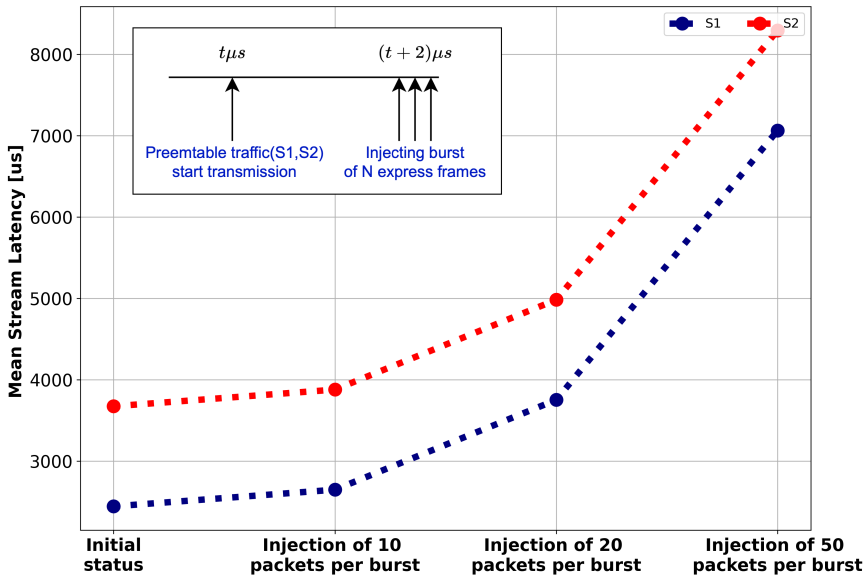


Figure 4.6: Injecting express traffic to delay specific preemptable flow(s). ©2023 IEEE.

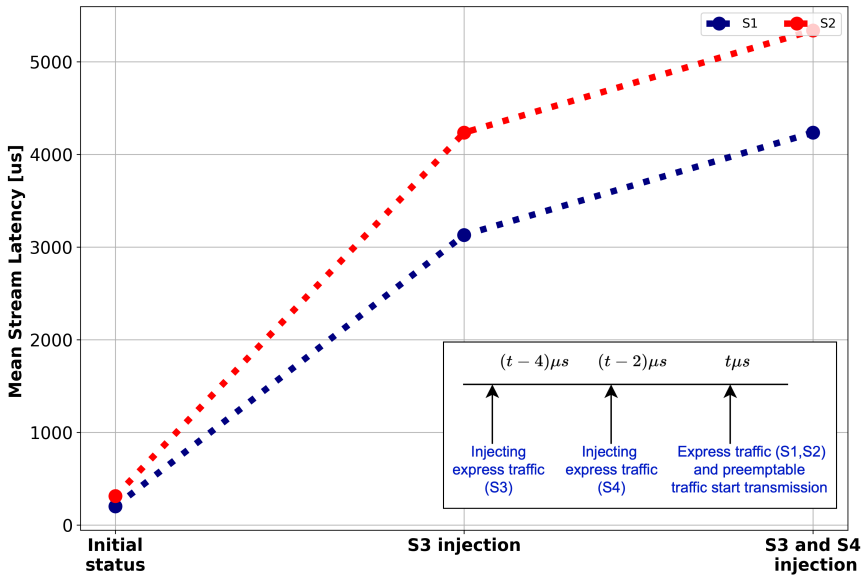


Figure 4.7: Injecting express traffic to delay specific express flow(s). ©2023 IEEE.

end-to-end latency of the express flows S1 and S2, around $250\mu s$, significantly increases with the injection of malicious traffic flows S3 and S4. In the worst case, this may lead to a violation of QoS constraints in the deadlines, resulting in packet loss.

Discussion and Implications

Frame preemption is one of the mechanisms employed within TSN to bound latency and ensure deterministic communication. This means that the latencies stick to the given upper bounds, but at the same time, this increases the predictability of the network. However, it is worth noting that this predictability can lead to notable degradation in QoS and also opens the possibility for preemptive DoS attacks. The findings confirm that based on the passive observation of a time-sensitive network, e.g., monitoring a single input and output port of a switch, can be exploited by malicious attackers.

Developing additional strategies that restrict the monitoring of the attacked network by attackers is beneficial to address this issue. One effective approach is introducing more dynamicity into the network, such as traffic routing dynamicity. Implementing dynamic routing paths within the network can make it harder for attackers to predict the current routing path of the flow. This complicates identifying patterns and predicting future events, even with an increasing attack budget regarding longer reconnaissance time or more powerful computational resources [ESKF23]. Here, the objective is to make it challenging for attackers to ascertain the forwarding of a particular flow by a specific TSN node or determine its processing location at any given time. These efforts align with our overarching goal of improving the resilience of time-sensitive networks against

DoS attacks and answers RQ5.

4.3 Dynamic and Scalable DoS Attack Detection and Filtering

This section introduces an approach for extracting an attack signature for describing the DoS attack patterns and filtering such attacks locally at the switch or network-wide through collaborative efforts. It addresses RQ6.

In the previous sections, we introduce attacks that violate their resource limitations and degrade the network performance, which may have more severe outcomes for TSN. Nevertheless, this is not the only scenario threatening the network's resilient operation. In certain situations, end hosts may transmit traffic in alignment with their resource reservations, making them appear legitimate and evading detection by filtering mechanisms. However, when end hosts collude, this strategy can lead to a DoS situation for the network

In such scenarios, the first step is extracting the attack signature that successfully differentiates attack and legitimate traffic. After that, this signature is used for mitigating these attacks and keeping the malicious traffic away from the network. However, filtering DDoS traffic is challenging as it originates from multiple distributed sources. To address that, content delivery networks employ special hardware to be distributed across the edge; however, this may still struggle to handle extremely large-scale DDoS due to the limited processing capacity [LM21]. Using rate limiting with hardware appliances can be effective in some scenarios, but it also has limitations, such as limited scalability and typically relying on pre-configured rules and policies. Software-based or hybrid solutions offer better adaptability, scalability, and control than hardware-based DDoS mitigation solutions [SM23]. As a result, the following additional questions emerge:

- RQ6.1: How can targeted attacks such as DDoS be accurately identified and mitigated in near-real-time while ensuring legitimate traffic is not impacted?

Accordingly, the following publication introduces a methodology for attack signature extraction by examining only the packet header and a proposed collaborative approach for attack mitigation. While it may not be specifically tailored for TSN, the centralized network controller of TSN makes it suitable for such a collaborative attack filtering [3]:

> Nurefşan Sertbaş Bülbül and M. Fischer. SDN/NFV-based DDoS Mitigation via Pushback. IEEE International Conference on Communications (ICC), 2020.

[3]For the given publication, the whole contribution belongs to this thesis. The co-author helped to improve the quality of the paper with his valuable feedback.

In the rest of this section, we present the details of our local and collaborative attack mitigation strategies. The respective publication is also attached in Appendix F.

Methodology

DDoS traffic cannot be easily filtered, as it comes from distributed sources. That highlights the importance of finding good representations of the attack traffic that differentiates the legitimate traffic. Even if the attack pattern is very well described, filtering that traffic on-premise still overloads the local firewall with the fixed processing capacity. To address that, we revise the *pushback* mechanism introduced in [MBF+02] to mitigate the impact of a DDoS by informing upstream network devices to block malicious traffic. We leverage SDN/NFV to position the filtering functionality dynamically in the network. It has two main building blocks as following:

i. **Extraction of the attack signatures via attribute-oriented induction**

Finding good representations of the attack patterns, namely the attack signatures, is essential for effective attack mitigation. For this purpose, we use attribute-oriented induction (AOI), a generalization technique to capture attack behavior that is usually spread across multiple flows and from distributed sources, e.g., DDoS attacks. Unlike approaches based on machine learning that generalize data on a tuple-by-tuple basis, AOI employs attribute-by-attribute generalization. This approach is highly effective for generating high-level representations.

Our AOI based signature extraction method does not require deep packet inspection; it only works with the packet headers and checks a few attributes only. Since attackers can easily spoof IP addresses, we use autonomous system number (ASN)s. The selected tuples are source ASN *srcASN*, source port *srcPort*, destination ASN *dstASN*, destination port *dstPort*, and protocol *prot*. Then, these tuples are used by AOI to generalize the captured network traffic and create clusters of similar traffic flows. Here, different hierarchies can be used for the generalization step. In this study, we use a range for the port numbers, AS membership for the IP addresses, and TCP/UDP-based generalization. Following the generalization, the pattern of the biggest cluster is selected as the attack signature. Then, depending on the environment, this signature is used for filtering the malicious attack traffic. For instance, the IPtables rules in Figure 4.8 b can be easily derived from the extracted signature in Figure 4.8 a. Since the actual values of the attributes such as IP addresses and port numbers that we use to extract the attack signature are known; we can reuse them in the generated IPtable rule accordingly. In the example, we replace the ASN with the related IP addresses, ip1,ip2, and ip3.

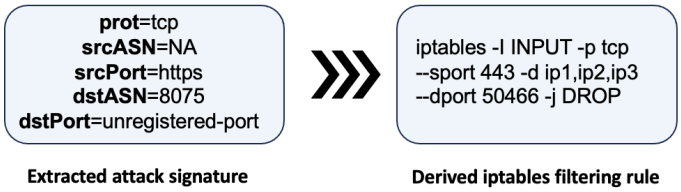


Figure 4.8: An example iptables filtering rule from the extracted attack signature. ©2020 IEEE.

ii. **Mitigation of the attacks via pushback mechanism:** We use the *pushback* mechanism to mitigate identified attacks efficiently and protect networks from severe congestion as a result of a rapid increase in traffic, e.g., from a DoS attack. The benefit of this mechanism is apparent in large scenarios in which the attacker aims to block the communication between local networks or between the network controller and entities within the network. The overall system illustration is given in Figure 4.9. Traffic received by an SDN domain contains both the malicious traffic (from R2 and R5) as well as the legitimate traffic (from R5). Once the attack has been detected, traffic will be analyzed further to generate attack patterns so as not to drop legitimate traffic, e.g., coming along R5. Thus, the centralized SDN controller monitors the traffic continuously based on the reports from the switch at the ingress of the SDN domain, S3. When the controller detects any attack pattern, this is used to create an open flow protocol (OFP) rule, and S3 is configured with this rule to filter out the traffic that fits this pattern.

   If the attack surpasses the capacity of S3, the pushback mechanism is triggered, initiating collaborative attack mitigation. The derived rule is propagated from R1 to R2, R3, and R5. Informed routers begin filtering traffic, ensuring the at-

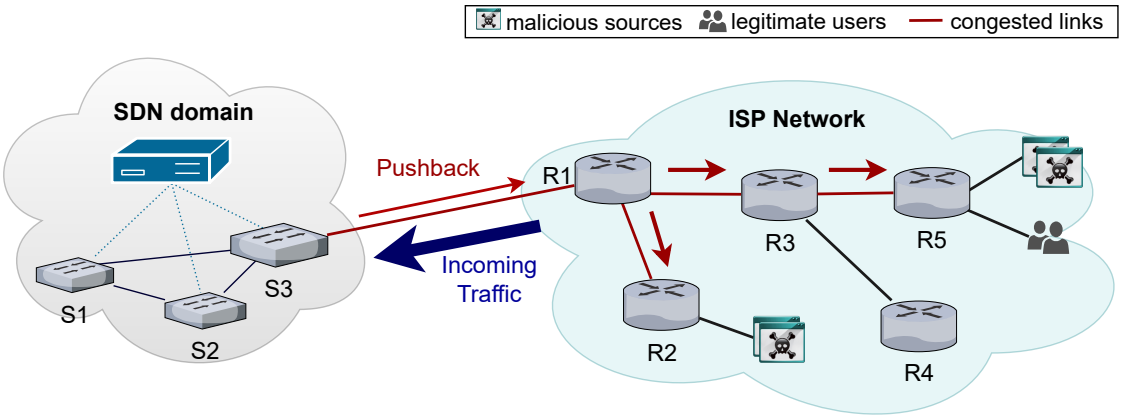


Figure 4.9: Attack mitigation with pushback mechanism. ©2020 IEEE.

tack traffic is filtered out before reaching the victim. In certain scenarios, such as collaborating with a non-SDN network, we can introduce this filtering functionality by initializing VNF. This collaborative mitigation approach aims to preserve network resources for legitimate users by restricting the spread of attack traffic.

The pushback mechanism works for traditional and hybrid networks such as TSN. The difference here may be that pushback filters at the network layer, while TSN operates at the link layer. In practice, combining these two layer filtering functions would be useful, as the link layer filtering controls the admission to the local network, and network layer filtering controls traffic between different networks or segments.

Key Results

We evaluate AOI based performance in extracting attack signatures on different real-world datasets such as CICDDoS [SLHG19], CICIDS [SLG18], ISCXIDS [SSTG12] and CTU-13 [GGSZ14]. To demonstrate the effectiveness of AOI in distinguishing attacks regardless of the attack type or dataset, we evaluate the classification rate (CR), false positive rate (FPR), and recall values. Additionally, we compare AOI with one approach from the related work that uses longest matching prefix (LMP) approach that internet routers use to choose a forwarding entry from a routing table. In this setup, LMP refers to sorting the destination IP addresses of the dropped packets by prefixes and defining the attack signature as the most frequent prefix.

Accordingly, the evaluation results are summarized in Table 4.1. Except for the CICDDoS dataset, AOI performs better than LMP regarding all metrics. FPR clearly shows that LMP fails to differentiate attack and legitimate traffic. For the CICDDoS dataset, the LMP approach demonstrates a slightly better CR compared to AOI. However, LMP fails to differentiate between legitimate and attack traffic effectively. While LMP drops an additional 2% of attack traffic compared to AOI (difference between the CRs), it also discards 39.41% of legitimate traffic (FPR), whereas AOI only drops 4.73%. This high rate of dropping legitimate traffic results from the dataset's source and destination IP distribution. As only one IP is under attack, LMP decides to drop all traffic directed to

	Attribute Oriented Induction			**Longest Matching Prefix**		
Dataset	**CR**	**Recall**	**FPR**	**CR**	**Recall**	**FPR**
CICIDS	0.9955	1.0	0.012	0.6900	1.0	0.8456
CTU-13	0.9988	0.9995	0.018	0.7063	0.7265	0.7467
CICDDoS	0.9747	0.9748	0.0473	0.9979	0.9991	0.3941
ISCXIDS	0.9036	0.9958	0.2270	0.7471	0.9958	0.6056

Table 4.1: Evaluation of AOI and LMP algorithms with varying attack types. ©2020 IEEE.

that IP, including legitimate traffic. However, AOI employs five attributes for filtering, resulting in the preservation of 95.27% of legitimate traffic from being dropped. Thus, in scenarios like this, when a single server is targeted instead of a complete network, the attack signature generated by the LMP matches all traffic designated to the server and drops the legitimate traffic.

For further enhancements on the AOI and processing overhead, e.g., CPU consumption, please refer to our original publication [SF20] (Appendix F).

👉 Discussion and Implications

Section 4.1 presents a proactive defense approach operating at the link layer with a fixed network position and capacity, `TSN Gatekeeper`. However, in certain scenarios, there may be a need to dynamically position security features across the network where they are most needed. In such cases, utilizing SDN/NFV-based pushback mechanism becomes crucial. These mechanisms aim to handle attacks locally and then collaboratively push back the filtering rules to filter attack traffic close to their origin. The pushback mechanism operates at the network layer and fits hybrid networks well. Moreover, it helps protect network resources from being wasted and enhances the QoS. Although we have not specifically tested this mechanism within a TSN setting, its generic nature suggests it could offer substantial benefits to such networks, albeit at a different layer.

Our findings indicate that extracting attack signatures and filtering based on them can effectively mitigate DDoS.

Chapter 5

Conclusion

Mission-critical networks serve as the backbone for various industries, from manufacturing and transportation to healthcare and telecommunications. They can host non-critical and critical traffic flows that require stringent quality of service (QoS). While increasing heterogeneity and connectivity render them more vulnerable to failures and attacks, conventional networking paradigms cannot satisfy these requirements. Therefore, time-sensitive networking (TSN) emerged to enable the coexistence of traffic of different criticality with different QoS requirements in the same network. Moreover, TSN is based on standardized Ethernet equipment enhanced with time-sensitive features. However, the current standardization of TSN cannot fully support new use cases that demand more dynamicity, heterogeneity, and complexity. Therefore, in this thesis, we address this gap by presenting a few solutions for the autonomous configuration and resilient operation of time-sensitive networks, which are not addressed yet in the standards.

Accordingly, in this *cumulative* thesis, we propose more flexible configuration solutions to enable time-sensitive networks to adapt to changing traffic more. Furthermore, we introduce methods for increasing the resilience of time-sensitive networks against denial of service (DoS). We first define several research questions regarding the self-adaptive and efficient configuration of time-sensitive networks related to transparent resource reservation (`RQ1`), dynamic network reconfiguration (`RQ2`), and machine learning (ML)-based low-cost network configuration (`RQ3`). Then, we define our research questions regarding the resilient operation of time-sensitive networks related to admission control (`RQ4`), the feasibility of calibrated attacks leveraging network monitoring (`RQ5`) and lastly, DDoS attack detection and collaborative filtering (`RQ6`). Then, we address those questions by presenting the details of our six contributions (identified with `C`) throughout the six research articles. Our contributions aligned with the respective research questions are summarized.

Self-Adaptive and Efficient Configuration of Time Sensitive Networks: Our contributions to self-adaptive and efficient configuration of time-sensitive networks consists of (`C1`) a dynamic self-configuration framework for removing the end-host-related dependencies, (`C2`) reconfiguration strategies for better resource utilization, and (`C3`) machine learning-based methods for the intelligent configuration.

> RQ1: How to design an autonomous configuration solution for seamless TSN resource reservation without requiring end hosts to be TSN-aware, while ensuring the desired QoS and reducing the configuration overhead?

C1. The configuration of TSN requires TSN-aware end hosts and their active participation in the resource reservation negotiation process. Accordingly, for transparent resource reservation in TSN, we propose a self-configuration framework in [SEF21] (Appendix A), SC-TSN, that follows the plug-and-play nature of Ethernet networks. The framework removes the end-host-related dependencies of TSN and enhances the switches at the edge with the traffic characteristics extraction modules. This way, we can automatically extract traffic resource requirements and put flow to a suitable routing path that satisfies its individual QoS. SC-TSN provides a time-sensitive delivery guarantee without the prior knowledge of flows, with only a slight increase in the average frame delay rate (less than 1%).

> RQ2: How to dynamically reconfigure time-sensitive networks at runtime to efficiently use network resources considering reconfiguration overhead? Is it feasible to do this in real time?

C2. In standard TSN deployments, routing paths are typically configured when an application is initiated and remain static. This can lead to suboptimal assignments of flows when flows change. This results in some links/switches being underutilized while others may become overloaded. Accordingly, we propose different flow placement heuristics in [SEF22] (Appendix B). These heuristics are based on the optimization problem presented in our previous work [SEF21] and offer reconfiguration solutions for different environments, assessing the reconfiguration overhead and the characteristics of the environment. Unlike reconfiguring the network for every new flow request, which is impractical for this environment, proposed heuristics can decrease the configuration time up to three times while accommodating more flows.

> RQ3: How can ML-driven strategies be leveraged to achieve efficient resource reservations in TSN? How close can these strategies approximate optimal performance?

C3. Finding multiple routing paths in TSN with different QoS requirements and limited network resources is challenging. Therefore, we propose an approach based on reinforcement learning (RL) that interacts with the network environment and learns

the optimal resource allocations [SF22] (Appendix C). We design our reward function considering the individual flow deadlines. With our reinforcement learning-based routing approach, we can decrease the computational overhead significantly compared to computing the optimal solution, e.g., via optimization problem. Our evaluation results indicate that our approach, `RL`, can satisfy QoS requirements for 96.12% of all frames even for the high link utilizations, which perform close ($\approx$ 1.5 %) to the optimal solution with a low computation cost.

Resilience against Denial of Service Attacks for Time Sensitive Networks: Our contributions to the resilience against denial of service attacks for time-sensitive networks include (`C4`) deploying admission control strategies to enforce flow reservations (`C5`), investigating the impacts of calibrated attacks and countermeasures, and (`C6`) dynamic and scalable denial of service (DoS) attack detection and collaborative filtering.

> `RQ4: What types of admission control strategies could effectively enforce flow reservations, and what would be the associated deployment costs?`

`C4.` TSN requires a precise configuration of switches. Any violation of a configuration might directly impact the performance of the network. Thus, detecting deviations, e.g., due to compromised or faulty network entities, and mitigating them immediately is important. For that, we propose a programming protocol-independent packet processors (P4)-based admission control strategies via in-network attack filtering in [SKF23] (Appendix D) to defend the network directly on the data plane. Employed filtering strategies can decrease the frame loss rate of the legitimate flows significantly, e.g., $\approx$24% in meter-blocking and $\approx$48% in gate-thresholding strategies.

> `RQ5: What information can an attacker gain from passive monitoring of TSN traffic, and is this information exploited for attacking time-sensitive networks? Moreover, what countermeasures can be used to protect the network against such malicious activities?`

`C5.` The determinism of TSN mechanisms ensures that data transmission and communication operate predictably within predefined time bounds. This determinism is crucial for guaranteeing the reliability and precision required in applications such as industrial automation and real-time control systems. Attackers might use this to capture traffic characteristics and launch tailored DoS attacks on the network to deteriorate the QoS. Considering this, we investigate the standard frame preemption

mechanism and demonstrate DoS attacks in [SF23] (Appendix E). Our simulation results indicate that, depending on the scenario, even a single attacker can cause significant delays for specific traffic classes and degrade the QoS of the network. Furthermore, we introduce countermeasures against such attacks, e.g., randomizing a few parameters in the network to invalidate/limit the attacker's knowledge.

> RQ6: How can malicious and legitimate traffic distinguished, and how this be leveraged to eliminate or minimize the impact of attacks in near-real-time?

C6. In the proactive filtering approaches deployed at fixed positions with a fixed processing capacity, filtering rules are pre-determined and remain static. This lack of adaptability makes the network vulnerable to DoS attacks. Thus, in [SF20] (Appendix F), we utilize attribute-oriented induction (AOI) approach for extracting signatures of DoS attacks on the fly. Our results indicate that AOI can extract signatures with a very high recall value of 97.48% and result in a classification rate of more than 90.36% even in the worst case. Filtering rules can be directly applied locally or pushed to the upstream networks of malicious traffic, enabling a collaborative approach to tackle DoS. This solution works on the network layer and could complement the filtering solution at the link layer as presented in C4 for TSN.

In conclusion, this thesis focuses on the self-adaptive and efficient configuration of time-sensitive networks and highlights the importance of resource optimization and QoS management in modern network environments. Furthermore, our efforts related to the resilience against DoS attacks uncover a limited number of potential DoS scenarios that might be critical for such networks. Lastly, we believe that integrating efficient configuration and resilience mechanisms to counter DoS attacks plays a significant role in shaping the future of time-sensitive networks, enabling them to meet the demands of an increasingly interconnected world.

Future Work

This thesis contributes novel solutions to self-adaptive and efficient configuration and resilient operation of TSN. However, we see further potential for future work in the following areas:

TSN in future (mobile) networks: As TSN will further evolve and contribute to be adopted in mission-critical applications, their seamless integration with existing networks, including legacy systems and emerging technologies like 5G and IIoT, becomes

necessary. Such integration brings new possibilities for enhanced communication and coordination across diverse network infrastructures, improving overall system efficiency and responsiveness. Thus, investigating the integration of TSN into wireless technologies such as 5G is valuable future research.

Exploring distributed mechanisms in TSN: Centralized network control mechanisms within TSN have proven their effectiveness in ensuring deterministic and reliable communication. These mechanisms are very useful in scenarios in which a global network view is beneficial. The central controller can make more informed decisions regarding resource allocation, traffic prioritization, and QoS. However, as a centralized controller is a single point of failure, distributed approaches scale better and are more fault-tolerant. There is a lack of hybrid models that leverage the strengths of both central and distributed mechanisms, thereby achieving a balance between scalability, determinism, and adaptability. This approach would be particularly valuable in large-scale and dynamic network environments. Therefore, deploying hybrid solutions by shifting some functions from the centralized controller to switches is an interesting future work.

Integration of digital twin concept in TSN research: Digital twins are virtual replicas of physical systems or processes and provide a real-time, data-driven representation of their behavior. Thus, integrating this concept into TSN enables researchers to conduct complex simulations in a risk-free environment with real data. They can be used to simulate the impact of policies before implementing them into the actual network, so it helps in making informed decisions and avoiding unintended consequences. Thus, it contributes to the efficiency and resilience of TSN and maintaining QoS. Therefore, building a digital twin to asses possible outcomes of the management decisions or predict the future behavior of the network stands as valuable future research areas.

Supporting future networking scenarios: The future of dynamic networking scenarios for TSN may require reshaping how we manage and optimize critical communication systems. The fusion of TSN technology with 5G networks and edge computing induces new use case scenarios requiring dynamic network management due to more dynamic traffic. The self-configuration framework presented in this thesis is the first step to support more dynamic traffic scenarios. Thus, future work is needed to include more dynamic scenarios, such as dynamic period changes on the fly. Also, not only the detection of the new periods but also how to handle these new demands, e.g., by relocating the flow or sending the flow over multiple paths, is an interesting research question.

Use of ML for optimizing TSN: TSN often operates in dynamic and complex environments where it is often hard to uncover patterns, dependencies, and correlations. Thus, ML-based methods have significant potential to be used in TSN due to their ability

to analyze, predict, and adapt to complex network conditions in real-time. The reinforcement learning-based routing solution presented in this thesis has promising results regarding QoS-aware resource allocation. Moreover, it has further potential use cases in TSN such as predicting the network traffic patterns or future demands to optimize QoS, optimizing packet scheduling algorithms to minimize the packet delays, or analyzing network traffic patterns to identify suspicious activities. Thus, by leveraging machine learning, time-sensitive networks can better meet the stringent requirements of critical applications in diverse domains.

Flow (re)placement strategies for TSN: Several scenarios, such as failures and dynamic traffic patterns, might require the replacement of the flows in the network while maintaining the timing requirements. As a part of this thesis, we introduce a few heuristics and asses their impacts on the TSN performance. Alternatively, scheduling flows proactively by anticipating future network demands or considering the possible replacement cost in the replacement decision process are engaging strategies for future research.

Investigating the calibrated attacks with varying attacker budget: In the calibrated attacks presented in this thesis, we assume the worst-case attacker has some network knowledge, such as network topology. Here, assessing the feasibility of attacks referring to different levels of information and capabilities that an attacker possesses is an interesting future research. For instance, whether an attacker with less knowledge can exploit the frame preemption mechanism to attack the network needs further investigation. Moreover, other TSN standards regarding their resilience against such calibrated attacks might also need to be assessed.

Bibliography

[AHG21] Anna Arestova, Kai-Steffen Jens Hielscher, and Reinhard German. Simulative evaluation of the TSN mechanisms time-aware shaper and frame preemption and their suitability for industrial use cases. In *2021 IFIP Networking Conference (IFIP Networking)*, pages 1–6. IEEE, 2021.

[ASS19] Abdullah Alnajim, Seyedmohammad Salehi, and Chien-Chung Shen. Incremental Path-Selection and Scheduling for Time-Sensitive Networks. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.

[ESKF23] Doganalp Ergenc, Florian Schneider, Peter Kling, and Mathias Fischer. Moving Target Defense for Service-Oriented Mission-Critical Networks. In *2023 32nd International Conference on Computer Communications and Networks (ICCCN)*, pages 1–10, 2023.

[FDR18] Jonathan Falk, Frank Dürr, and Kurt Rothermel. Exploring practical limitations of joint routing and scheduling for TSN with ILP. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 136–146. IEEE, 2018.

[GGSZ14] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123, 2014.

[Gin21] Ginthör, David and Guillaume, René and Nayak, Naresh and von Hoyningen-Huene, Johannes. *Time-Sensitive Networking for Industrial Control Networks*. Springer International Publishing, 2021.

[HFG+20] David Hellmanns, Jonathan Falk, Alexander Glavackij, René Hummen, Stephan Kehrer, and Frank Dürr. On the Performance of Stream-based, Class-based Time-aware Shaping and Frame Preemption in TSN. In *2020 IEEE International Conference on Industrial Technology (ICIT)*, pages 298–303, 2020.

[IEE10] IEEE 802.1 TSN Task Group. IEEE Standard for Local and Metropolitan Area Networks–Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP). *IEEE Std 802.1Qat-2010*, pages 1–119, 2010.

[IEE16a] IEEE 802.1 TSN Task Group. IEEE Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic. *IEEE Std 802.3br-2016* , pages 1–58, 2016.

[IEE16b] IEEE 802.1 TSN Task Group. IEEE Standard for Local and Metropolitan Area Networks — Bridges and Bridged Networks - Amendment 24: Path Control and Reservation. *IEEE Std 802.1Qca-2015*, pages 1–120, 2016.

[IEE16c] IEEE 802.1 TSN Task Group. IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption. *IEEE Std 802.1Qbu-2016*, pages 1–52, 2016.

[IEE16d] IEEE 802.1 TSN Task Group. IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic. *IEEE Std 802.1Qbv-2015*, pages 1–57, 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–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing. *IEEE Std 802.1Qci-2017*, pages 1–65, 2017.

[IEE18] IEEE 802.1 TSN Task Group. IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements. *IEEE Std 802.1Qcc-2018*, pages 1–208, 2018.

[LDH+10] Zhenhui Li, Bolin Ding, Jiawei Han, Roland Kays, and Peter Nye. Mining Periodic Behaviors for Moving Objects. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, page 1099–1108, New York, NY, USA, 2010. Association for Computing Machinery.

[LM21] Zihao Li and Weizhi Meng. Mind the amplification: cracking content delivery networks via DDoS attacks. In *Wireless Algorithms, Systems, and Applications: 16th International Conference, WASA 2021, Nanjing, China, June 25–27, 2021, Proceedings, Part II 16*, pages 186–197. Springer, 2021.

[LWF+21] Feng Luo, Bowen Wang, Zihao Fang, Zhenyu Yang, and Yifan Jiang. Security Analysis of the TSN Backbone Architecture and Anomaly Detection System Design Based on IEEE 802.1 Qci. *Security and Communication Networks*, 2021.

[MBF+02] Ratul Mahajan, Steven M Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Aggregate-based congestion control. *Computer Communication Review*, 32(3), 2002.

[MHKS19] Philipp Meyer, Timo Häckel, Franz Korf, and Thomas C Schmidt. DoS Protection through Credit Based Metering - Simulation-Based Evaluation for Time-Sensitive Networking in Cars. *Proceedings of the 6th International OMNeT++ Community Summit*, 2019.

[NDR16] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. Time-Sensitive Software-Defined Network (TSSDN) for Real-Time Applications. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, page 193–202, New York, NY, USA, 2016. Association for Computing Machinery.

[NDR17] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. Incremental flow scheduling and routing in Time-Sensitive Software-defined Networks. *IEEE Transactions on Industrial Informatics*, 14(5):2066–2075, 2017.

[OYN20] Mubarak Adetunji Ojewale, Patrick Meumeu Yomsi, and Borislav Nikolić. Multi-level preemption in TSN: feasibility and requirements analysis. In *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*, pages 47–55. IEEE, 2020.

[SLG18] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. 2018.

[SLHG19] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani. Developing Realistic DDoS Attack Dataset and Taxonomy. IEEE, 2019.

[SM23] Denis Salopek and Miljenko Mikuc. Enhancing Mitigation of Volumetric DDoS Attacks: A Hybrid FPGA/Software Filtering Datapath. *Sensors*, 23(17), 2023.

[SSTG12] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374, 2012.

[VYC05] Michail Vlachos, Philip Yu, and Vittorio Castelli. On periodicity detection and structural periodic similarity. In *International conference on data mining*. SIAM, 2005.

[WC92] Zheng Wang and Jon Crowcroft. Analysis of shortest-path routing algorithms in a dynamic network environment. *ACM SIGCOMM Computer Communication Review*, 22(2):63–71, 1992.

[YCW22] Zihan Yu, Suzhi Cao, and Xin Wang. An Online Incremental Scheduling Method for Time Sensitive Networks Combined with Routing. In *2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP)*, pages 934–939, 2022.

[ZWY+21] Yao Zheng, Shuo Wang, Shuwen Yin, Binwei Wu, and Yunjie Liu. Mix-flow scheduling for concurrent multipath transmission in time-sensitive networking. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2021.

Acronyms

AGV automated guided vehicles.

AI artificial intelligence.

AOI attribute-oriented induction.

AS autonomous system.

ASN autonomous system number.

BE best-effort.

CNC centralized network configuration.

CPU central processing unit.

CR classification rate.

CUC centralized user configuration.

DDoS distributed denial of service.

DoS denial of service.

ECT equal cost tree.

FEF fill empty first.

FPR false positive rate.

GCL gate control list.

IIoT industrial internet of things.

ILP integer linear programming.

IoT internet of things.

IP internet protocol.

IS-IS intermediate station to intermediate station.

IT information technology.

LMP longest matching prefix.

MCN mission-critical network.

MILP mixed integer linear programming.

ML machine learning.

MSTI multiple spanning tree instance.

NFV network function virtualization.

OFP open flow protocol.

OT operations technology.

P4 programming protocol-independent packet processors.

PCE path computation element.

PCR path control and reservation.

PSA portable switch architecture.

PSFP 802.1Qci per stream filtering and policing.

QoS quality of service.

RL reinforcement learning.

SDN software-defined networking.

SPB shortest path bridging.

SRP IEEE 802.1Qat stream reservation protocol.

TAS IEEE 802.1Qbv time-aware shaper.

TCP transport control protocol.

TDMA time division multiple access.

TSN time-sensitive networking.

TSOR time sensitive optimal routing.

TT time-triggered.

UDP user datagram protocol.

VNF virtual network function.

Appendices

Copyright Notice

This appendix presents publications as originally published and reprinted with permission from the corresponding publishers. The copyright of the original publications is held by the respective copyright holders; see the following copyright notices.

Appendix A

Paper 1: 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

Nurefşan Sertbaş Bülbül, D. Ergenç, M. Fischer. SDN-based Self-Configuration for Time-Sensitive IoT Networks. IEEE 46th Conference on Local Computer Networks (LCN), 2021. ©2021 IEEE.

Contribution

In the forementioned publication, the contributions of this thesis are designing the whole framework, implementing a simulation model, and evaluating the overall proposal. The second co-author modeled and implemented the optimization model (`TSOR`) and its complexity analysis and integration into the self-configuration framework. The third co-author helped improve the paper's quality 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

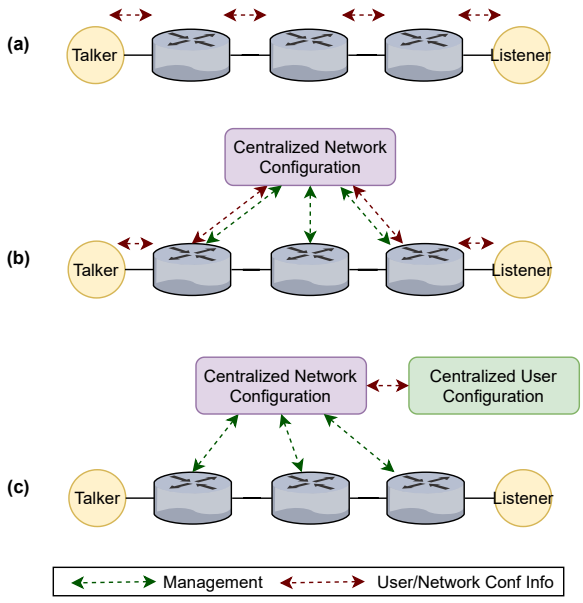


Fig. 1: TSN configuration models.

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.

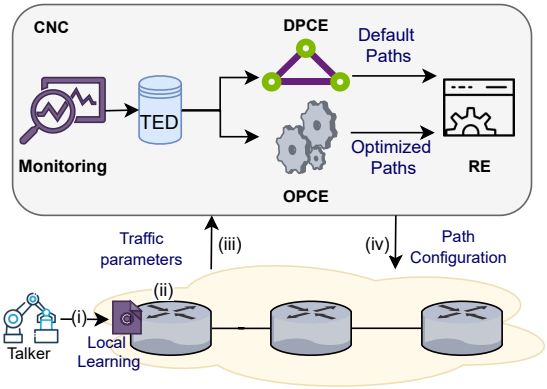


Fig. 2: Overall system architecture.

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 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.

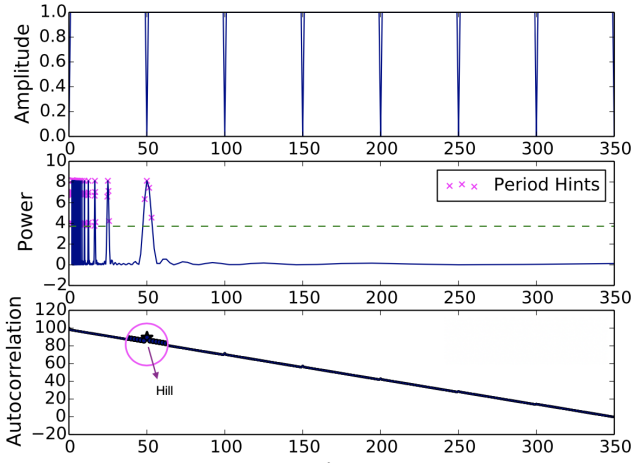


Fig. 3: Period extraction for a sample sequence.

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.

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

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.

$$\sum_{d \in D} \sum_{p \in P_d} x_{dp} \alpha_{ep} h_d \leq c_e \qquad \forall e \in E \tag{2}$$

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. $\alpha_{ep} = 1$.

$$\sum_{s \in S} g_{es} = 1 \qquad \forall e \in E \tag{3}$$

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}.

$$\sum_{p \in P_d} \sum_{e \in E} x_{dp} \alpha_{ep} \left[l_e^o + l_e^q (1 - g_{es}) \right] \leq l_d \qquad \forall d \in D \tag{4}$$

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.

$$g_{es} - \sum_{d \in D} \sum_{p \in P_d} x_{dp} \alpha_{ep} \frac{h_d}{c_e} \geq 0 \qquad \forall e \in E, \forall s \in S \qquad (5)$$

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.

$$x_{dp} \geq a_{dp} \qquad \forall d \in D, \forall p \in P_d \qquad (6)$$

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.

$$\min \sum_{d \in D} \sum_{p \in P_d} \sum_{e \in E} x_{dp} \alpha_{ep} \left[l_e^o + l_e^q (1 - g_{es}) \right] \qquad (7)$$

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

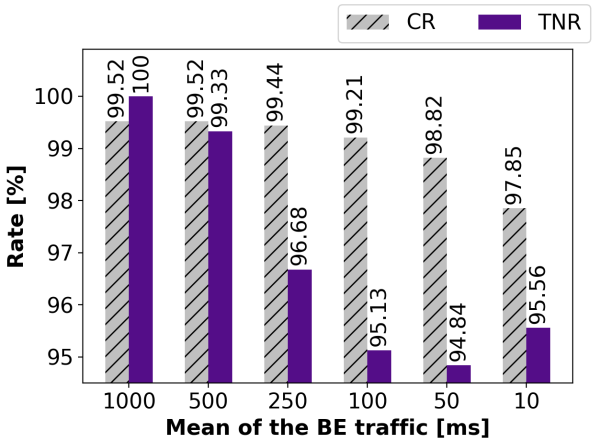


Fig. 4: Classification performance of the learning module.

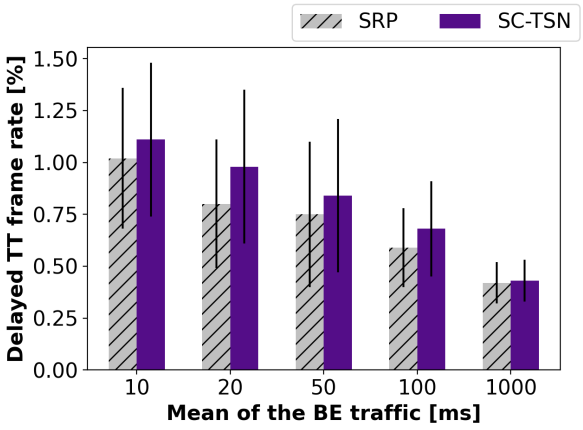


Fig. 5: Delayed TT frame rate.

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

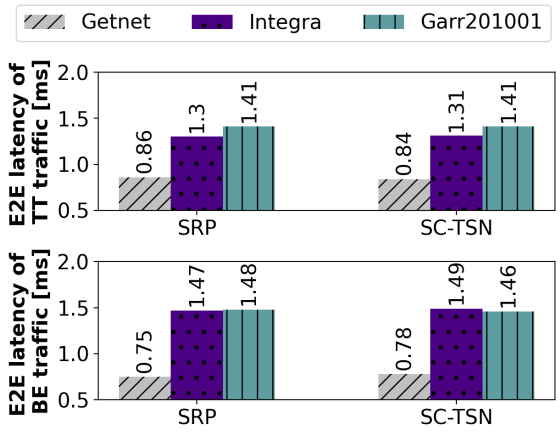


Fig. 6: Performance comparison for varied sized topologies.

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.

guarantee bounded latency. However, the configuration of gate

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 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 B

Paper 2: 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

Nurefşan Sertbaş Bülbül, D. Ergenç, M. Fischer. Towards SDN-based Dynamic Path Reconfiguration for Time-sensitive Networking. IEEE/IFIP Network Operations and Management Symposium, 2022. ©2022 IEEE.

Contribution

In the forementioned publication, the contribution of this thesis is to propose the overall idea, implement the variations in a simulation model, and conduct the evaluation. The second co-author designed and implemented the restricted and unrestricted versions of the optimization model (`TSOR`). The third co-author helped improve the paper's quality 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

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.

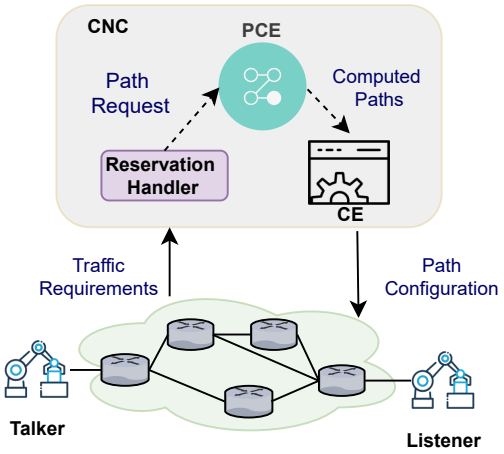


Fig. 1. Overall system block diagram

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

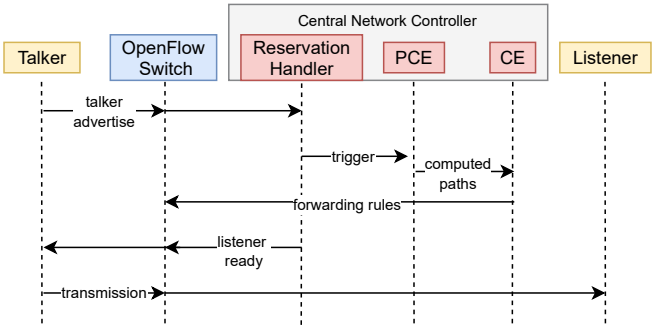


Fig. 2. Time sequence of the overall system

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

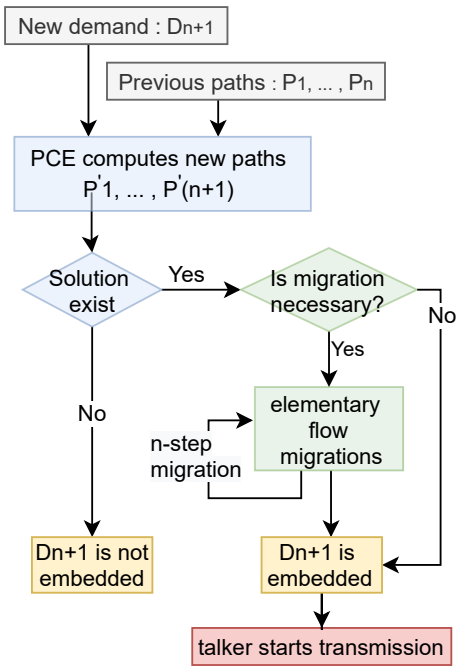


Fig. 3. Flowchart for the path configuration

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 endpoints. 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

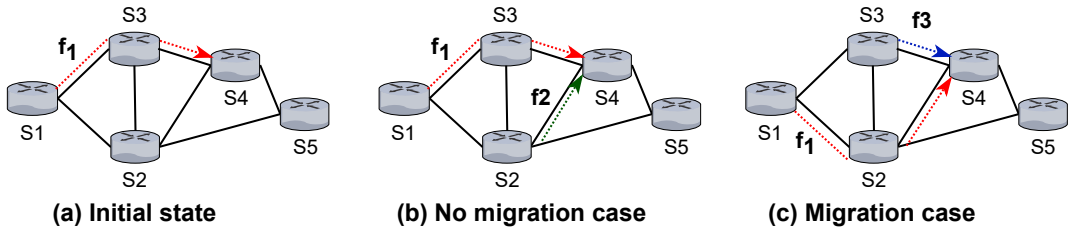


Fig. 4. Example scenario

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.

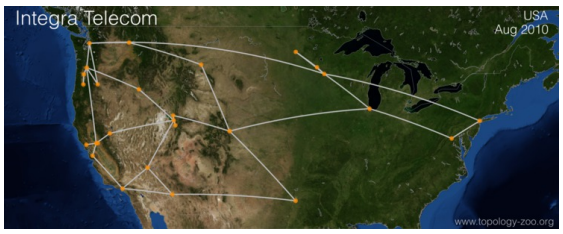


Fig. 5. The Integra topology used in the evaluation [16]

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

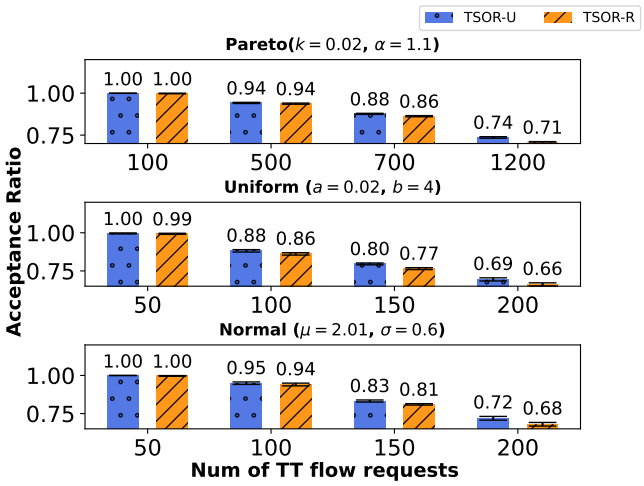


Fig. 6. Acceptance ratio with varying data rate distributions

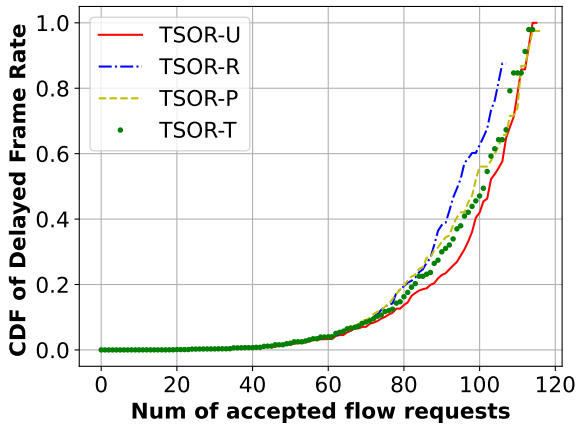


Fig. 7. Missing deadline ratio respect to number of accepted flows

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.

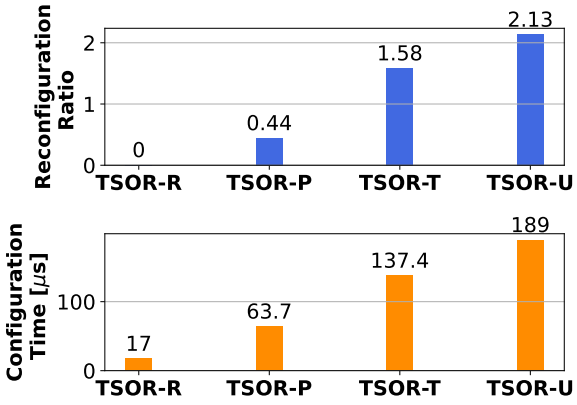


Fig. 8. Reconfiguration overhead for TSOR

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 C

Paper 3: Reinforcement Learning assisted Routing for Time Sensitive Networks

Abstract

Recent developments in real-time critical systems pave the way for different application scenarios such as Industrial IoT with various quality-of-service (QoS) requirements. The most critical common feature of such applications is that they are sensitive to latency and jitter. Thus, it is desired to perform flow placements strategically considering application requirements due to limited resource availability. In this paper, path computation for time-sensitive networks is investigated while satisfying individual end-to-end delay requirements of critical traffic. The problem is formulated as a mixed-integer linear program (MILP) which is NP-hard with exponentially increasing computational complexity as the network size expands. To solve the MILP with high efficiency, we propose a reinforcement learning (RL) algorithm that learns the best routing policy by continuously interacting with the network environment. The proposed learning algorithm determines the variable action set at each decision-making state and captures different execution times of the actions. The reward function in the proposed algorithm is carefully designed for meeting individual flow deadlines. Simulation results indicate that the proposed reinforcement learning algorithm can produce near-optimal flow allocations (close by $\sim 1.5\%$) and scales well even with large topology sizes.

Reference

Nurefşan Sertbaş Bülbül and M. Fischer. Reinforcement Learning assisted Routing for Time Sensitive Networks. IEEE Global Communications Conference (GLOBECOM), 2022. ©2022 IEEE.

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.

Reinforcement Learning assisted Routing for Time Sensitive Networks

Nurefşan Sertbaş Bülbül and Mathias Fischer
University of Hamburg, Germany
Email:{sertbas, mfischer}@informatik.uni-hamburg.de

***Abstract*—Recent developments in real-time critical systems pave the way for different application scenarios such as Industrial IoT with various quality-of-service (QoS) requirements. The most critical common feature of such applications is that they are sensitive to latency and jitter. Thus, it is desired to perform flow placements strategically considering application requirements due to limited resource availability. In this paper, path computation for time-sensitive networks is investigated while satisfying individual end-to-end delay requirements of critical traffic. The problem is formulated as a mixed-integer linear program (MILP) which is NP-hard with exponentially increasing computational complexity as the network size expands. To solve the MILP with high efficiency, we propose a reinforcement learning (RL) algorithm that learns the best routing policy by continuously interacting with the network environment. The proposed learning algorithm determines the variable action set at each decision-making state and captures different execution times of the actions. The reward function in the proposed algorithm is carefully designed for meeting individual flow deadlines. Simulation results indicate that the proposed reinforcement learning algorithm can produce near-optimal flow allocations (close by $\sim 1.5\%$) and scales well even with large topology sizes.**

***Index Terms*—delay aware routing, TSN, reinforcement learning, resource allocation, routing optimization**

I. Introduction

With the rapid advancement of communication technologies future industrial networks are expected to support a large number of Internet-of-Things (IoT) devices with a wide range of QoS needs. Critical applications, e.g., remote robotic surgery, requires guaranteed data transfer with bounded low latency, low delay fluctuations, and no data loss. Thus, low-latency and deterministic networking is a prerequisite for real-time IoT. The IEEE 802.1 working group has suggested time-sensitive networking (TSN) standards to address the real-time and deterministic communication requirements of time-sensitive and mission-critical systems. TSN is an Ethernet extension that allows mission-critical applications with varying QoS requirements to take advantage of Ethernet's high bandwidth and cheap commercially available off-the-shelf (COTS) hardware.

Diverse QoS requirements and the exponentially increasing TSN network traffic make it difficult to manage networks [1]. More specifically, sending time-critical traffic over routes that preserve individual QoS requirements of the flows becomes more challenging. Several solutions were proposed by researchers to allocate network resources (e.g., link bandwidth) efficiently under resource constraints. Using a shortest-path routing algorithm is not suitable as serious congestion and increased latencies that violate strict flow deadlines might be the result. Besides, its slow convergence speed is not suitable for dynamic networks. For that, many researchers formulate this problem as a mixed-integer linear program, which is NP-hard and its computational complexity increases exponentially increasing as the network size expands [2], [3].

As an alternative, reinforcement learning has attracted much attention from researchers due to its intelligent decision-making and large-scale applicability. It has obvious advantages in routing applications such as high throughput and low latency and enables adaptive routing. Such features make it a promising solution for time-critical networks by reducing system costs in conjunction with meeting user expectations. Moreover, reinforcement learning also supports the self-configuration concept for TSN and it can be used as a black-box optimization in continuous time.

The main contribution of this paper is the application of reinforcement learning to the problem of optimal routing of time-sensitive flows considering individual flow deadlines. Accordingly, our contributions are:

- We propose a reinforcement learning-based routing method that determines the best routing path for various scenarios by adaptively evaluating paths depending on the flow deadline constraints.
- We evaluate our approach via realistic OMNeT++ simulations and compare it with benchmarking algorithms regarding the runtime and time-sensitive traffic delivery performance. The simulation results indicate that our RL-based routing approach scales well with the increasing topology sizes and still produces nearly-optimal flow allocations.

The remainder of this paper is structured as follows: Section II summarizes related work on reinforcement learning-based routing approaches. Section III describes the basics of the Q-learning algorithm. 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

Reinforcement learning has been already applied to finding efficient paths in the networking domain. Depending on the environment, several QoS metrics have been used to define the reward function such as available link bandwidth, link delay,

packet loss ratio [4], [5], deadline [6]. In [7], the authors use reinforcement learning to find efficient paths that maximize the network throughput and minimize communication delay in software-defined networks (SDN). However, the solution requires deploying one agent per switch.

The most commonly used method in reinforcement learning applications is Q-learning which is known for its learning efficiency and applicability to various problems. In [8], the authors propose a delay-based reward mechanism to update link weights and find the next switch to forward the packet. Even though their experiments show that the Q-learning-based strategy selects paths with low less delay, their approach is not applicable to the TSN environment and also generates additional probe packets to poll the current network status. The authors of [6], proposed a Q-learning approach named safe RL for the real-time routing problem. For that, they defined edges with their average expected and worst-case delays that are determined by the pre-processing phase. Then, each node, locally, decides on the next node to forward by checking whether the deadline of the packet will be satisfied or not. However, they train their model using a fixed deadline, so that the generated Q-table only works for a fixed deadline which is not realistic for such an environment. Instead, they would need to train RL for every possible deadline to obtain their individual Q-tables, which is impractical. Another problem is the trustworthiness of the worst-case bounds and knowledge of the distribution of the average delay of the edges. To address these issues, the authors in [9] propose a table-driven algorithm assuming a priori knowledge of the exact delay probability distributions of the edges. Moreover, they also propose an RL-based approach in case such an estimation is unknown or dynamic. Unlike [6], their Q-learning implementation does not require training for every possible deadline. They reward every action, which is from one node to another, and update Q-values directly.

However, such hop-by-hop approaches are not directly applicable to TSN, since TSN needs to ensure a maximum number of hops and end-to-end latency for flows. Here the solution can be either redesigning the reward mechanism by considering the hop count or taking the action set as a set of end-to-end paths instead of hops. We define an action as a selection of a path from among the possible paths that ensure the maximum hop count and reward as a function of meeting the flow's deadline. The presented design is then evaluated using realistic TSN simulations.

III. Reinforcement Learning Model: Q-Learning

Reinforcement learning is a sort of machine learning algorithm in which agents are rewarded for their actions. Q-learning which was first introduced in [10], is a model-free reinforcement learning method based on the Markov decision process (MDP), but unlike MDP, it's possible to learn without knowing anything about the environment beforehand.

For that, the agent takes feedback from the environment, called a reward, and uses that feedback to evaluate the prior actions. So, a reward function indicates what is good (or bad) in an immediate sense. The aim of the RL agent is to maximize the long-term reward. The Q-learning algorithm is usually modeled by three tuples as S, A, R, where S is a set of states, A is a set of actions, and R is a reward function. Here, every decision that an agent takes is called as *action*. With the taken action agent moves from the current state s_t to a new state, s_{t+1}. The state-to-state transition denotes a movement from one state to the next, with the action being the selection of the next best state. More formally, Q-learning is based on the action-value pair $Q(s, a)$ where $s \in S$, and $a \in A$ where S is a set of states and A is a set of actions. Each action is rewarded afterward by the reward function R and it is desired to select an action that delivers the most benefit (e.g., reduced latency) in the current system conditions.

During the learning process, a policy (Q-table) is maintained and the entries (Q-values) in the Q-table are updated iteratively by the following equation

$$\begin{aligned} Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t) \\ + \alpha[R(s_t, a_t) + \gamma max_{a_{t+1}Q(s_{t+1}, a_{t+1})}] \end{aligned} \tag{1}$$

where α is a learning rate that shows how much the model will learn new values vs the old values. The γ is the discount factor that balances immediate and future rewards. Both α and γ values takes real numbers in the range of 0 to 1. $Q(s_t, a_t)$ is the Q-value at time t and shows the expected reward for the given state-action pair, s_t and a_t. Here, the Q-value is not directly assigned but rather updated using a mechanism similar to gradient descent depending on the learning rate which can be realized as a decaying function. As a result, by selecting an appropriate learning rate, α, $Q(s_t, a_t)$ can converge to an ideal value.

IV. Q-Learning based Time-Sensitive Routing

In this section, we introduce our RL-based routing solution for TSNs. We first describe the overall framework, and afterward, we explain our routing solution in detail.

A. System Overview

We propose an RL-based routing framework for TSN by combining it with SDN and a central network controller. The global view of a centralized SDN controller enables to use of centralized routing algorithms and thus eases the configuration. Routing paths can be reconfigured dynamically considering the requirements of the time-sensitive flows and corresponding reward values can be collected as feedback to the installed flow path.

We illustrate our SDN-based framework in Fig. 1. Here, the end-host, a talker in TSN, declares its flow traffic requirements, e.g., the deadline, by sending a talker-advertise message to the controller (i). Then, the stream registration module records the parameters extracted from the message and requests a path from the RL model that satisfies the latency requirement (ii). The RL model decides on a routing path and then the controller installs the required forwarding rules using the data plane configuration module (iii). Afterward, the controller informs the related talker by sending a listener-ready message which

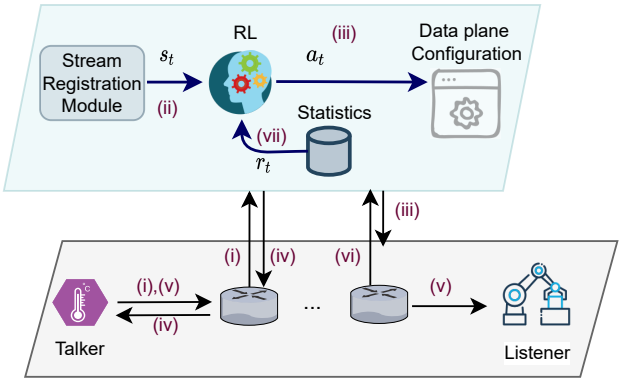


Fig. 1. Overall system block diagram

states that resources are allocated for the related transmission (iv). When the talker receives a listener-ready message, it will start streaming by sending frames via the switch close by and the data plane will forward frames to the listener (v). As a feedback mechanism, the end switch will report measured statistics such as latency to the controller (vi). These collected statistics are used to compute the reward value for the related action, and routing path, and update the RL model (vii).

B. Time sensitive Routing Problem

We used a reinforcement learning strategy based on Q-learning for delay-sensitive routing in the TSN environment because traffic conditions in dynamic networks may lead to updated routing paths based on traffic delay encountered at each switch and link in the network.

In our design, the agent is the SDN controller and it gets feedback from the data plane (e.g., TSN switches). A *state* is a traffic matrix that represents the current network load and an *action* taken by the controller is a selection of a path to route the new incoming flow. To eliminate the problems that may occur in hop-by-hop route selection, as stated in Section II, we define the selection of an end-to-end path as an action. In other words, we limit the set of possible actions to ensure to meet the maximum number of hops constraint of TSN and we use the reinforcement learning policy to decide which end-to-end path (action $a \in A$) to take from the current state s.

Algorithm 1: Q-learning based routing - control plane

Input: Learning rate: α
Discount factor: γ
Exploration and exploitation parameter: ϵ
List of paths: $\mathcal{P}$
Network link-states: $\mathcal{S}$

1: Initialize Q: Q(s,a)
2: **while** Path request received from src to dst with a deadline D **do**
3: Generate a random number m
4: **if** $m < \epsilon$ **then**
5: select random action, a = $\mathcal{P}[rand(0, numOfPaths)]$
6: **else**
7: select action, a = $Q_{max}(s, a)$
8: **end if**
9: Install forwarding rules for the selected action a (aka path)
10: **end while**
11: **while** Reward update message from the data plane **do**
12: Updates Q-table and moves a new network state:
13: $Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma max_{a_{t+1}Q(s_{t+1}, a_{t+1})}]$
14: **end while**
15: **return** Optimal routing policy

Algorithm 2: Q-learning based routing - data plane (e.g., switch S_{dst})

Input: Received frame, f, from src to dst with a deadline D
1: **while** f.destination == S_{dst} **do**
2: $t_{E2Elatency} = f.arrivalTime - f.creationTime$
3: $\Delta t = D - t_{E2Elatency}$
4: $R = \begin{cases} 1 + \frac{\Delta t}{D}, & \text{if } \Delta t \geq 0 \\ \frac{\Delta t}{D}, & \text{otherwise} \end{cases}$
5: Computed reward, R, is sent to the controller
6: **end while**

The aim here is to learn the optimal routing policy that maximizes the accumulated reward over time. For that, it is needed to define a reward function that guides the policy to the desired behaviour, i.e., routing while preserving individual flow deadlines in our environment. Thus, unlike the previous studies, we define our reward function that is received at the end of successful frame reception based on deadline satisfaction instead of latency. The reward, R is obtained after the successful reception of a frame and is calculated as follows:

$$R = \begin{cases} 1 + \frac{\Delta t}{D}, & \text{if } \Delta t \geq 0 \\ \frac{\Delta t}{D}, & \text{otherwise} \end{cases} \quad (2)$$

where D is a flow deadline and Δt is the remaining time until the deadline, which can be computed by taking the difference between end-to-end latency, $t_{E2Elatency}$, of the frame and its deadline. Here, the reward values for the actions (e.g., selected path) can be either positive or negative. For the non-negative Δt values, we define the reward by dividing Δt by the deadline D so that, flows that have smaller deadlines will get a greater reward for the same Δt values. Additionally, to give a positive reward in case the deadline is met, even though Δt is zero, we add 1. For the negative Δt values that represent the missed deadlines, we use Δt proportional to D as a penalty which is also a negative number. After a time, negative rewarded actions are slowly going to be filtered out, while actions that lead to a positive reward are going to appear more frequently. Therefore, we will be able to choose paths for flows that offer a higher deadline satisfaction rate for newly registered flows.

Algorithm 1 shows the pseudo-code of our algorithm that runs on the SDN controller. The aim is here to develop a policy to select a path to send a packet from origin src to destination dst. Initially, when there is a path request, the ϵ-greedy policy chooses one action from the vector of feasible actions (Line 3-8). Here, the algorithm explores the feasible action space with a probability ϵ at the same time while a search for a

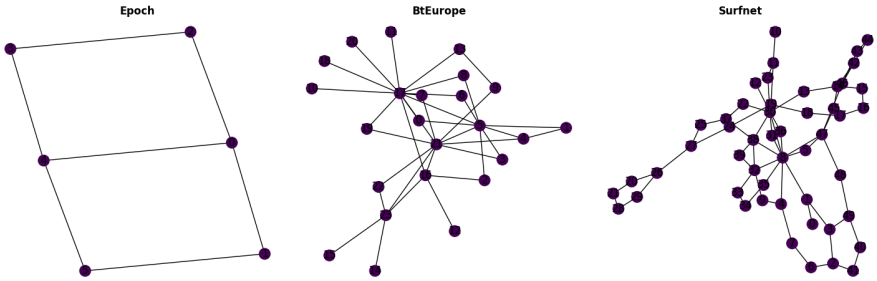


Fig. 2. Real world evaluation topologies; small-scale(Epoch), medium-scale(BtEurope), large-scale(Surfnet)

path that has a higher reward with a probability of $1-\epsilon$ since there is no pre-knowledge about the paths. The ϵ value usually selected a small number so that the policy may take advantage of knowledge about the present state the majority of the time. After the action selection, the SDN controller will enforce related forwarding rules to the data plane and the host will start the transmission (Line 9). When frames over this path are received by the destination switch S_{dst}, the reward for this transmission is computed as shown in Algorithm 2 and sent to the controller. The SDN controller uses obtained reward to update Q-table for the related path and makes use of this state-action value function to make a more informed decision (Line 11-14 in Algorithm 1).

V. Evaluation

In this section, we evaluate our RL-based routing model for time-sensitive traffic and compare the proposed approach against other benchmark algorithms. First, we briefly explain the evaluation setup and metrics. Then, we evaluate the performance of our RL-based routing model at varying traffic loads. Finally, we summarize our evaluation results.

A. Experimental Setup

We have simulated realistic TSN networks in OMNeT++ v5.5.1 using the INET and SDN4CoRE frameworks that enable the configuration of TSN switches via SDN [11]. With that setup, we evaluate the quality of the routing paths generated by the proposed RL model in terms of different metrics such as end-to-end latency, deadline satisfaction rate, jitter, and run time.

For our experiments, we used different real-world topologies, e.g., Epoch, BtEurope, and Surfnet, selected from the Topology Zoo dataset [12] as shown in Fig. 2. 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 sake of simplicity and set the simulation time to 40 seconds. Due to the lack of 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 [13]. For that, we generated isochronous and cyclic traffic as time-sensitive traffic. The isochronous traffic has the highest priority, its period changes uniformly between 500 us to 2 ms and the payload is set to 50 bytes. This type of traffic can be used for controller-to-controller and controller-to-I/O-communication where data must be produced and delivered consistently and where packets are delivered with bounded latency. We also generated cyclic traffic which still has high priority (e.g., 5 and 6), but is less critical than the isochronous traffic. The period of this type of traffic changes between 2-20 ms and the payload size changes between 50-1000 bytes. Example applications for this type of traffic may include input/output updates sent to/from actuators and sensors and a programmable logic controller in a manufacturing facility with short cycle times usually. We also generated best-effort (BE) traffic that has no assurance of timing and started our simulations with a larger number of existing BE flows in the network to measure the performance of the routing models when additional time-sensitive flows arrive. For our Q-learning model, we set the learning rate (α) to 0.7, the discount factor (γ) to 0.9, and the exploration parameter (ϵ) to 0.3. Table I summarize our simulation parameters.

B. Evaluation Metrics

We used the following metrics to evaluate our model:

TABLE I
Simulation Parameters

		Pattern	Period	Application Data Size	Priority
Traffic Characteristics	**Isochronous**	Periodic	0.5-2 ms	Fixed, 50 Bytes	7
	Cyclic	Periodic	2-10 ms	Fixed, 50-500 Bytes	6
		Periodic	11-20 ms	Fixed, 500-1000 Bytes	5
	Best-Effort	Sporadic	-	Variable, 30-1500 Bytes	2
		Num of switches	**Num of edges**	**Average node degree**	**Total number of nodes**
Topologies	**Epoch**	6	7	2.33	26
	BtEurope	24	37	3.08	119
	Surfnet	50	68	2.72	210

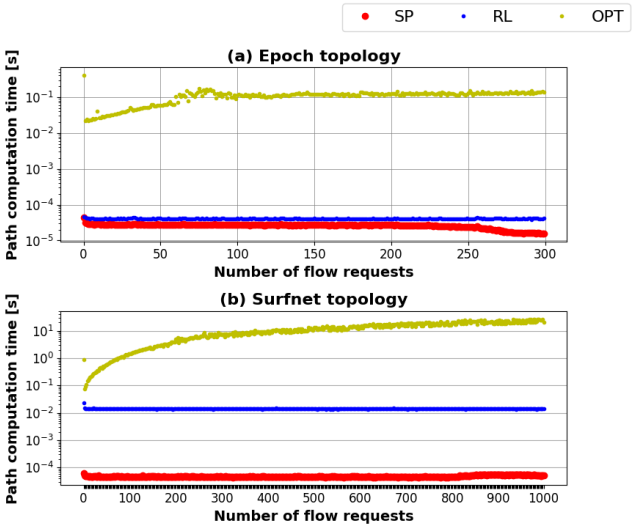


Fig. 3. Path computation time

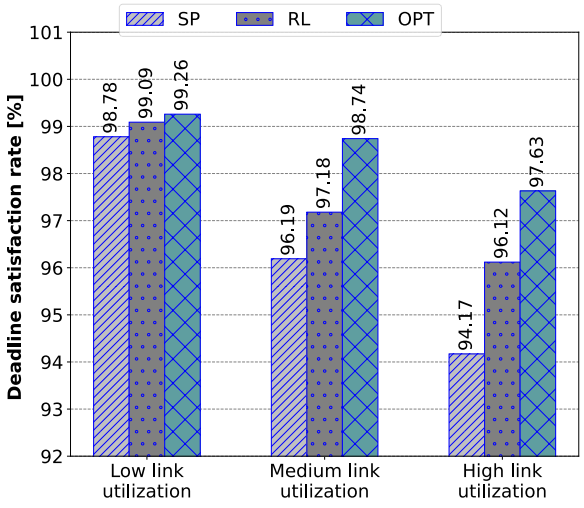


Fig. 4. Deadline satisfaction rate

- **(Incremental) path computation time:** It is the time required for path computation for n^{th} flow when there are $(n-1)$ flows in the network.
- **Deadline satisfaction rate (DSR):** It is the ratio of frames whose delay does meet its delay requirements.
- **End-to-end delay (E2E):** It is the end-to-end latency of frames until they reach their destination.
- **Jitter:** It is a variation of the E2E latency of frames.

C. Benchmarking Algorithms

a) Shortest Path (SP): It is the basic routing method, which directs data traffic via shortest paths, concentrating the data load on a few select links in the network. As a result, certain switch ports/links become congested. Higher contention means longer transmission latencies which might be a problem for the delivery of time-critical traffic. Since this will be the most straightforward solution with a minimal computation cost, we use *SP* as a lower bound for our approach to compare.

b) Optimal Routing via Integer Linear Programming (OPT): In [14], we introduced the time-sensitive optimal routing problem and a MILP formulation to find suitable paths that satisfy the end-to-end latency requirements of the demands. The objective is to minimize the overall latency of the selected paths considering the TSN-specific mechanisms such as gates at the egress of TSN switches. Since it will find the optimal resource allocation and routing path, we take it as an upper bound for the presented RL approach and measure how close RL gets to the optimum.

D. Results

This section compares our RL model with other benchmark algorithms in terms of presented evaluation metrics.

*a) **Path computation time**:* To evaluate the scalability of the algorithms, we used small-scale (e.g., Epoch) and large-scale (e.g., Surfnet) topologies and then measured path computation time independence on an increasing number of flow requests. The experiments were repeated 40 times for each scenario, and the results are given with a 95% confidence interval in Fig. 3. The x-axis represents the index of the current received flow request and the y-axis represents the time needed to embed the last received flow into the network on the logarithmic scale. Since both *SP* and *RL* are based on table-look-up, their computation time does not differ significantly even for large topologies such as Surfnet. However, while the pathfinding time of *OPT* is still acceptable in small topologies as in Fig. 3-a, it significantly increases when the topology gets larger as in Fig. 3-b. Thus, the results support our claim that *OPT* may not be suitable for large-scale scenarios due to its runtime complexity.

*b) **Time sensitive traffic delivery**:* To measure the delivery performance of our *RL* model and the benchmarking approaches for TSN, we simulated a realistic TSN network in OMNeT++. For these experiments, a medium-scale topology, that is BtEurope in our scenario, is used. We embedded best-effort flows initially to saturate the network resources. We also generate 180 time-triggered flows for low average link utilization ($\approx$ 15%), 450 time-triggered flows for medium average link utilization ($\approx$ 45%) and 900 time-triggered flows for high average link utilization ($\approx$ 75%). Then, we measured the deadline satisfaction rate of time-triggered traffic in dependence on varying link utilizations as shown in Fig. 4. As expected, the satisfaction rate decreases significantly in the high utilization case (e.g., from 99.26% to 97.63% in *OPT*) due to the already occupied resources. However, we see much considerable decrement in *SP*, from 98.78% to 94.17% due to the lack of ability to find better paths in case of congestion. For the *RL* algorithm, it can be said that it stays close to *OPT* with the 96.12% deadline satisfaction rate. In other words, only ($1 - DSR = 3.88\%$) of the frames do not meet their deadlines even for the higher utilization scenarios.

We also measure the deadline satisfaction rate of individual traffic classes (DSR-7, DSR-6, DSR-5), E2E latency, and jitter as given in Table II. Here, the reason behind the high deadline satisfaction rate in all algorithms (e.g., 100% DSR for priority

TABLE II
OMNeT SIMULATION RESULTS

	Low utilization			Medium utilization			High utilization		
	SP	**RL**	**OPT**	**SP**	**RL**	**OPT**	**SP**	**RL**	**OPT**
DSR-7	98.4 %	98.81 %	99.03 %	95.05 %	96.33 %	97.12 %	92.51 %	94.95 %	96.82 %
DSR-6	100 %	100 %	100 %	99.86 %	99.93 %	99.98 %	99.01 %	99.57 %	99.98 %
DSR-5	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %
E2E latency	260 μs	230 μs	210 μs	420 μs	340 μs	280 μs	600 μs	530 μs	340 μs
Jitter	180 μs	150 μs	140 μs	320 μs	260 μs	200 μs	410 μs	400 μs	250 μs

5 traffic) is that lower priority traffic has higher deadlines. For instance deadlines of the priority 5 traffic can take values between 11-20 ms while priority 7 traffic has much more strict deadlines between 0.5-2 ms (See Table I). Here, we see once again that while there is only 98.4% of DSR in low utilization, this decreases to 92.51% in the high utilization scenario for *SP*. It can also be seen that *RL* still stays close to *OPT*, e.g., 1,8% less deadline satisfaction then the *OPT* at worst.

In addition to the DSR, we also measure the E2E delay and jitter. Even though the results are still better than *SP* and close to *OPT*, the difference may not be that obvious. Because while designing our reward function, we chose to meet the deadlines instead of minimizing the delay and imposing a penalty when there are unsatisfied deadlines. For example, in a case in which the deadline is 5 ms, there is no difference in terms of *RL* in terms of sending this traffic via a 2 ms route versus sending it via a 4.9 ms route. Therefore, seeing E2E delays closer to *SP* compared to *OPT* can be explained in this way.

VI. Conclusion

In this paper, we studied the efficient path allocation problem for flows with individual QoS constraints. The problem can be solved by a mixed-integer linear program (MILP), which is NP-hard and becomes exponentially more complex as the network size grows, or with a straightforward routing such as the shortest path which results in congestion for highly utilized networks. This motivates us to employ reinforcement learning to obtain near-optimal solutions to route the time-sensitive flows with reduced computational complexity. Here, since supervised learning approaches require a priori knowledge about the network which is not feasible for our scenario, we believe reinforcement learning would be a perfect fit as it completely relies on the environment's feedback for its decisions. Our results show that the proposed approach achieves to satisfy deadlines with a rate of 94.95%, even in the worst case for the most critical traffic, which is close to the optimum and its run time complexity does not increase as the topology size increases (as in the MILP).

We believe that the presented approach in this study can work under the self-configuration concept which the networking community shifts towards. Thus, more research is needed to make the proposed technique more resilient by training across more diverse network conditions (e.g., failures and dynamic changes in the topology). As future work, we plan to extend our algorithm to also generate not only the routes but also the transmission schedules of TSN.

References

[1] Y. Seol, D. Hyeon, J. Min, M. Kim, and J. Paek, "Timely survey of time-sensitive networking: Past and future directions," *IEEE Access*, vol. 9, pp. 142 506–142 527, 2021.

[2] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "Ilp-based joint routing and scheduling for time-triggered networks," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, 2017, pp. 8–17.

[3] J. Falk, F. Dürr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for tsn with ilp," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 136–146.

[4] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, "Intelligent routing based on reinforcement learning for software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 870–881, 2021.

[5] A. A. Magadum, A. Ranjan, and D. G. Narayan, "Deepqosr: A deep reinforcement learning based qos-aware routing for software defined data center networks," in *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 2021, pp. 1–7.

[6] G. N. Seetanadi, K.-E. Årzén, and M. Maggio, "Adaptive routing with guaranteed delay bounds using safe reinforcement learning," in *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, 2020, pp. 149–160.

[7] Y.-R. Chen, A. Rezapour, W.-G. Tzeng, and S.-C. Tsai, "Rl-routing: An sdn routing algorithm based on deep reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 3185–3199, 2020.

[8] T. Mahboob, Y. R. Jung, and M. Y. Chung, "Optimized routing in software defined networks–a reinforcement learning approach," in *International Conference on Ubiquitous Information Management and Communication*. Springer, 2019, pp. 267–278.

[9] K. Agrawal, S. Baruah, Z. Guo, J. Li, and S. Vaidhun, "Hard-real-time routing in probabilistic graphs to minimize expected delay," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 63–75.

[10] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

Appendix D

Paper 4: TSN Gatekeeper: Enforcing Stream Reservations via P4-based In-network Filtering

Abstract

Real-time communication is crucial for mission critical scenarios, such as industrial automation and automotive applications. To meet these applications' strict quality of service (QoS) requirements, a new set of specifications, known as time- sensitive networking standards (TSN), has been proposed. TSN requires pre-registration of data streams before actual communication to help guarantee bandwidth and ensure constrained end- to-end latency. However, this mechanism is vulnerable to traffic overload and denial of service (DoS) attacks. This paper proposes a P4-based dynamic attack filtering as a link-layer network function to defend TSN against malicious network elements, such as faulty talkers or switches, directly on the data plane. Our experiments indicate that our P4-based implementation can filter malicious traffic with minimal overhead and minimize frame losses for legitimate traffic.

Reference

Nurefşan Sertbaş Bülbül, J.J. Krüger, M. Fischer. TSN Gatekeeper: Enforcing stream reservations via P4-based in-network filtering. The International Federation for Information Processing (IFIP) Networking Conference, 2023.

Contribution

For the given publication, the main contribution belongs to this thesis. The second co-author implemented and evaluated several parts of TSN Gatekeeper in the context of a master's thesis. The third co-author helped to improve the quality of the paper with his valuable feedback.

TSN Gatekeeper: Enforcing stream reservations via P4-based in-network filtering

Nurefşan Sertbaş Bülbül, Joshua Jannis Krüger and Mathias Fischer
University of Hamburg, Germany
Email:{nurefsan.sertbas, joshua.krueger, mathias.fischer}@uni-hamburg.de

***Abstract*—Real-time communication is crucial for mission critical scenarios, such as industrial automation and automotive applications. To meet these applications' strict quality of service (QoS) requirements, a new set of specifications, known as time-sensitive networking standards (TSN), has been proposed. TSN requires pre-registration of data streams before actual communication to help guarantee bandwidth and ensure constrained end-to-end latency. However, this mechanism is vulnerable to traffic overload and denial of service (DoS) attacks. This paper proposes a P4-based dynamic attack filtering as a link-layer network function to defend TSN against malicious network elements, such as faulty talkers or switches, directly on the data plane. Our experiments indicate that our P4-based implementation can filter malicious traffic with minimal overhead and minimize frame losses for legitimate traffic.**

***Index Terms*—time-sensitive networks, programmable data planes, ingress filtering, per-stream policing, babbling idiots**

I. Introduction

Ethernet is a family of standards that enables high-throughput communication in diverse wired-networking environments. It has also been proposed for use in modern embedded environments with real-time requirements, such as industrial and in-car networks [2]. However, these new application domains have introduced new requirements that Ethernet was not initially prepared to satisfy. For instance, self-driving cars with hundreds of time-sensitive sensors create new challenges for the underlying Ethernet architecture, such as timely delivery and bounded latency guarantees. To address these challenges, the IEEE time-sensitive networking task group has proposed a set of standards that can be used when the application has no tolerance for frame loss due to congestion and guaranteed upper bounds on end-to-end latency [3]. These standards also allow the coexistence of traffic with different priorities, such as high-priority time-critical and low-priority best-effort traffic, to be sent over the same physical infrastructure.

Time-sensitive networking relies on end hosts' pre-registration of traffic requirements to allocate the necessary resources along the end-to-end path. With this, the end host agrees with the network before the actual communication, and thus the network can provide a certain level of QoS for the requested end host, the talker in TSN. Every entity in TSN is expected to obey its reserved resource limits, such as staying within the allocated bandwidth. However, this mechanism is vulnerable to malicious network elements, such as faulty talkers or switches. A malicious talker may send more traffic than it previously reserved, which may cause congestion at the switches on the path and can violate bandwidth guarantees on all streams, even the legitimate ones. This phenomenon is called the *babbling idiot*, and whether it is intentional or not, it must be avoided to maintain determinism in such a network.

To overcome this issue, the IEEE TSN Task group has proposed the IEEE 802.1Q Qci Per-Stream Filtering and Policing (PSFP) standard [1]. The standard introduces a cascaded filtering mechanism that blocks or limits excessive amounts of data to protect queues from DoS attacks. Moreover, it enables the application of fine-grained policing decisions. However, the filtering approach has yet to be explicitly defined; it is only conceptually defined.

In this paper, to dynamically enable such filtering functionality within the network at line rate, we leverage network programmability via P4 language, which has recently attracted attention from both the research community and the industry [4]. P4 enables the implementation of novel network functions for various use cases, such as fine-grained packet handling and advanced packet forwarding, even without a centralized controller. It can also handle packets at line rate dynamically and flexibly. This makes it possible for network designers to create fine-grained networks aware of the applications and data being delivered, allowing the network to meet mission-critical requirements such as latency assurances [5]. Thus, due to such benefits, P4 has a good potential to accelerate innovations in time-sensitive networks [6]. Accordingly, our contributions are:

- We present two ingress filtering mechanisms that comply with the concepts introduced in the IEEE 802.1 Qci standard to safeguard switch queues against DoS attacks. We leverage P4 to implement the proposed filtering functionality directly at the data plane, eliminating the need for a centralized controller
- We evaluate the effectiveness of our approach by conducting experiments using randomly generated network topologies. We compared the performance of our proposed filtering mechanisms with the scenario where no filtering policy is deployed. Our evaluation results indicate that our filtering approaches incur minimal overhead, even with an increasing number of attackers, and effectively mitigate the impact of DoS attacks on switch queues and prevent the loss of legitimate traffic.

The remainder of this paper is structured as follows: Section

II describes basic TSN mechanisms and summarizes the state of the art. In Section III, we introduce our overall architecture. We evaluate our approach and describe our evaluation results in Section IV. Finally, Section V concludes the paper and summarizes future work.

II. Background and Related Work

A. Time Sensitive Networking

In a typical time-sensitive network, the sender of the data, a talker, announces the desire to send data by sending a talker-advertise message that defines the traffic characteristics of the stream. In this context, a stream refers to a data flow between the sender and receiver, such as the talker and listener(s) in TSN, and is identified by a unique stream identifier (StreamId). The talker-advertise message is propagated over the network depending on the configuration scheme, either centralized or distributed. All listeners receive the message, and only the listener(s) interested in receiving the related stream replies. The reply message, listener-ready, is forwarded in the reverse direction of the talker-advertise message back to the talker. On a listener-ready message switches on the path, re-check whether the resources to guarantee fault-free transmission are available. If so, the resources are reserved, and the listener-ready message is forwarded to the next switch. Eventually, the listener-ready message reaches the talker, which initiates the process, and the stream transmission can begin. If a stream is no longer needed, the talker and listener can cancel the associated resources by sending a cancel message [7].

With this pre-configuration, required resources, e.g., bandwidth, are reserved before communication. So that network can provide a specific latency guarantee for the related transmission. Here it has been assumed that every TSN node obeys its reserved resource limits, e.g., stays within allocated bandwidth. However, when there is an unauthorized attempt to use resources, e.g., DoS attacks by flooding, the network may no longer provide the promised QoS. To deal with traffic exceeding its pre-defined limits, a threshold-enforcing can be applied, in which traffic up to the advertised limit is forwarded, and any exceeding traffic is blocked. It does have a clear benefit in certain situations where a temporarily faulty end-host, which returns to normal operations after a period of violation, could be kept in the network while effectively containing its threats. Alternatively, the stream can be entirely blocked when it exceeds the advertised limit. Unlike the first one, all traffic is dropped at the ingress, even if the stream would later returns to behave as advertised.

The IEEE task group has proposed IEEE 802.1 Qci standard for ingress filtering traffic that exceeds its registered bandwidth guarantees. This filtering standard helps maintain the established service quality for specified traffic and streams, protect queues from unwarranted traffic (e.g., deliberate DoS attacks), and mitigate the effects of bandwidth violations and malfunctioning. The standard proposes a cascaded layer of filtering and policing as shown in Fig. 1. In the first layer, stream filters decide which gates and meters will be responsible for the arriving frame. In other words, when a frame arrives at the switch's ingress, the stream filter matches the StreamId to a specific, though not necessarily dedicated, stream gate and flow meter for policing and filtering. The second layer, a stream gate, is a two-state filter. In the *open* state, frames can pass to the responsible flow meter for further filtering. In the *closed* state, frames will be dropped. The state can change on a schedule which can be carried out by defining the gate control list or based on interaction by the control plane. Lastly, the flow meters enable the deployment of more fine-grained algorithms and decide whether the frame is allowed to pass. After the flow meter allows a frame, it gets queued in the network node for remaining forwarding or processing. This filtering mechanism supports the following policing actions [8]:

- Time-based policing: This can be carried out using stream gates as it has two states: *open* and *closed*. Frames that arrive when the gate is closed are directly filtered (discarded) so that this mechanism aims to support ap-

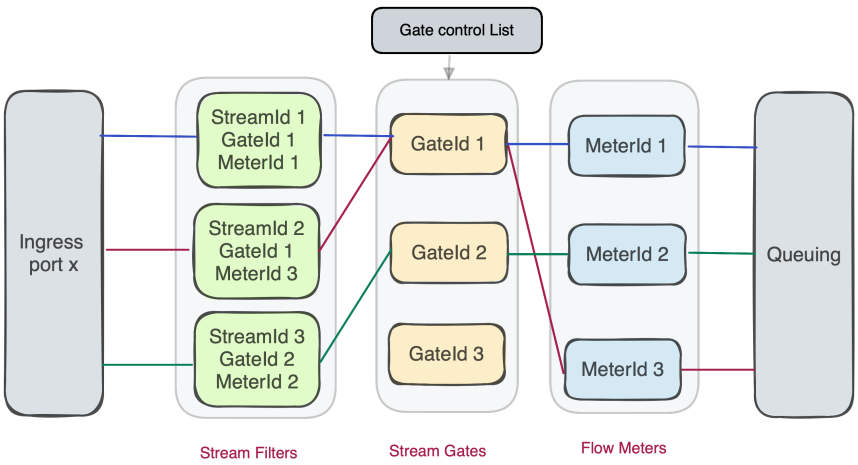


Fig. 1: IEEE 802.1Qci per-stream filtering and policing [1]

plications where the transmission and reception of frames across the network are coordinated.
- Rate-based policing: This can be carried out using flow meters by specifying frame rate parameters. Then, the meters apply to stream(s) and allow policing of streams that exceed the configured rate.
- Burst-based policing: This can be carried out using flow meters so that the length of the supported burst is set, and frames will be filtered accordingly.
- Frame length-based policing: This can be carried out using flow meters to filter frames based on the maximum frame lengths.

Authors in [9], propose an IEEE 802.1Qci-based attack detection system that applies filtering based on bandwidth and arrival time. For that, they propose a two-rate, three-color marker-based mechanism that can successfully drop illegitimate traffic. However, the configuration parameters of the presented approach are assumed to be set at the beginning and remain the same during the network's lifetime. Such a static configuration could be a valid assumption for specific use cases, e.g., in-vehicle networks. Still, such a solution would only partially benefit from the protocol's capabilities. Also, only the periodic, time-triggered traffic has been taken into account; but it needs to be clarified how to derive the filtering rules for aperiodic traffic, e.g., burst traffic. In [10], a filtering mechanism based on one of the existing traffic shapers in TSN, Credit Based Shaper, is proposed, and its compatibility with the 802.1Qci filtering is shown via OMNeT++ simulations. However, feasibility analysis with other traffic shapers is left as a future work.

B. P4 based Security

Several studies exist in the literature to add such attack filtering functionality to the network, either using centralized or distributed configuration options. Here, software-defined networks (SDN) could be an option with the global network view of the centralized SDN controller [11]. However, a centralized control plane can easily be a bottleneck in many scenarios, especially for recovery and connectivity protection [12], [13]. Also, the communication delay between data and the control plane may limit the reaction time of the controller as well [14]. Thus, such an implementation suffers from centralized controller dependency and causes scalability problems. Besides, standard SDN-based solutions are able to configure only per-flow forwarding policies and do not allow per-packet priority enforcement. Thus, these shortcomings of the SDN further motivate researchers to use the P4-based data planes, which enable processing packets at a line rate and remove the need for a centralized controller. Moreover, unlike the OpenFlow protocol, which is used as a standard protocol between controller data plane communication in SDN, P4 enables the definition of more fine-grained packet fields and, therefore, deployment of more use case-specific security solutions.

P4 has been used frequently in the literature to bring security functionality to the data plane and reduce the experienced latency by removing the remote controller involvement [15]. Using P4-based data planes in blockchain architecture, authors in [14] show that several attacks on the blockchain, including DoS, can be discovered before transaction packets get to the control plane. In [16], P4-based data paths are used to minimize latency and add security monitoring functionality for industrial 5G networks. Their results show that latency can be decreased by around half. In [17], authors use P4 to improve the routers to filter router spoofing and man-in-the-middle attacks directly on the data plane. Authors in [18] use P4 to detect spoof and volumetric attacks close to the source in order to protect the network. Their approach is scalable and controller-independent, thanks to the P4.

The flexible configuration options of the P4-based switches, either centralized or distributed, make P4 a promising solution for mission-critical networks. Thus, we believe that combining the merits of IEEE 802.1Qci and P4 marks a significant step in protecting future time-sensitive networks.

III. P4-based Ingress Filtering

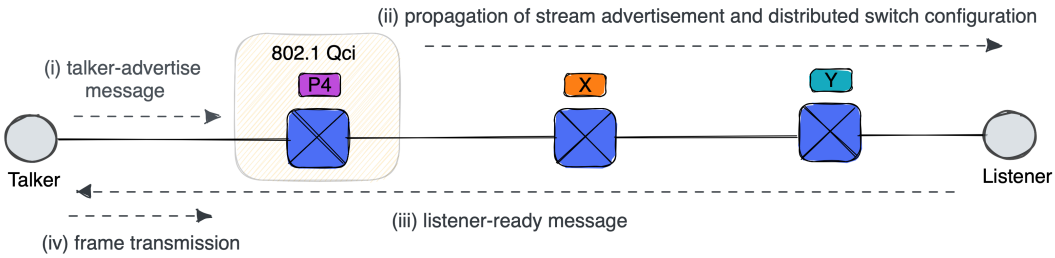
This section introduces our P4-based ingress filtering approaches for TSNs and how we dynamically deploy that strategy in time-sensitive networks. We first describe the overall frame filtering procedure, and afterward, we explain our filtering approaches in detail.

A. Overall System

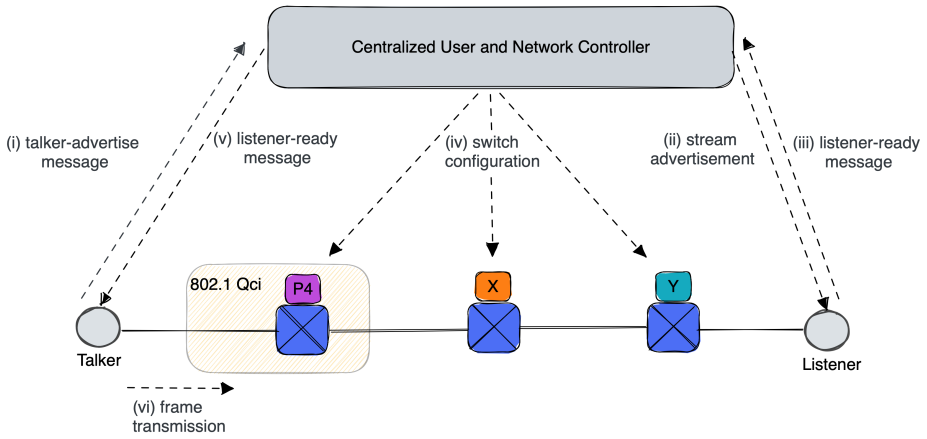
In time-sensitive networks, as explained in Section II-A, talkers must inform the network about the required resources before initiating transmission. Once the reservation is made, it is crucial to ensure that the talker complies with the declared traffic requirements. To achieve this, we propose a P4-based ingress filtering solution designed to run on the programmable data plane of a P4-enabled TSN switch. The solution uses P4 to implement a lightweight firewall at the edge of the TSN, which protects switches from being attacked or overloaded.

We envision the proposed P4-based filtering solution as a link-layer network function, as there could be other functions for different purposes, such as intrusion detection, load balancing, etc. (represented by X and Y in Fig. 2). The P4-based filtering solution can work independently at the edge switch without a centralized controller. However, it is still possible to configure P4-enabled TSN switches with different policies using a centralized network controller. We illustrate the overall architecture in Fig. 2 for both distributed and centralized configuration architectures, as our P4-based filtering approach works in both cases.

In the distributed architecture, a talker communicates with the edge switch to declare its traffic requirements (i), and the switch forwards the requirements to the other core switches in the network (ii). Here, switches are not configured by a central entity but in a distributed manner with their local knowledge. Our P4-based ingress filtering approach can also be configured as it derives filtering rules from the stream reservation messages. Then, interested listeners will subscribe to that stream (iii), and the talker will be informed to start transmission (iv). In the centralized architecture, the talker



(a) Distributed configuration architecture



(b) Centralized configuration architecture

Fig. 2: P4-based ingress filtering as a link-layer network function for TSNs

and listener communicate directly with the centralized user and network controller (i, ii). Then, the central controller sends switch configurations to the switches (iii). Here, the centralized controller can specify filtering rules and configure the ingress filtering module accordingly. Lastly, the talker is informed of the transmission, and then it starts stream transmission (v).

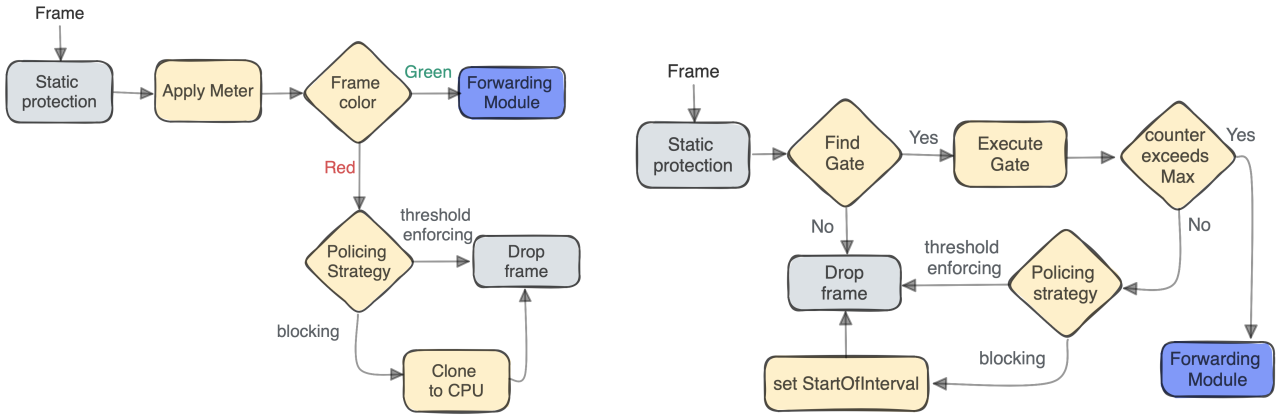
The presented filtering approach has two main blocks static and dynamic. We present a static mechanism as a first checkpoint, including maximum frame size and ingress port verification. Here, frames exceeding the maximum size are dropped to avoid possible switch congestion. Since attackers can still flood frames with the spoofed StreamIds to block or harm the transmission of other (legitimate) talkers, it is also necessary to verify the ingress port. After the initial verifications as a second checkpoint, we deployed our dynamic filtering solutions, namely *metered ingress filtering* and *gated ingress filtering*, as we describe in the following sections.

B. Metered Ingress Filtering

In the deployment of metered ingress filtering, we benefit from the portable switch architecture (PSA), which is a target architecture that defines standard data types, counters, meters, and other externs that P4 programmers can use as required. The P4 language design aims to maintain minimal consensus between switch vendors, excluding extended features. However, switch vendors can utilize architecture definitions to support more features and enable rapid innovation and proof of concepts before all parties accept them. Here, the PSA makes such P4 programs portable across different targets.

Based on the PSA primitive, we use direct meters derived from the RFC 2698 [19]. Conceptually, they are similar to the buckets and are defined by the initial burst size (BS). Then, the bucket size is increased by the pre-configured information rate (IR) times per second and decreased on the arrival of a packet. Here, the IR parameter can be computed based on the talker-advertise message as it represents the number of frames per measurement interval. The burst size can be interpreted as the number of frames by which a stream is allowed to exceed the advertised rate. Such a parameter would be helpful in case of frame delays that may normally require the frame, even the legitimate streams. If the bucket size falls below zero, packets are marked red, while otherwise, they are green. We use this mechanism for filtering by attaching a direct meter to the forwarding table, which is automatically executed in case a matching entry exists. The burst size and information rate parameters can be configured per table entry via the P4Runtime API, which conceptually stands as a control plane. Note that that is the control plane of the P4, not a centralized controller.

After applying the meter and marking the frame as red or green, frames are handled differently depending on the filtering



(a) Flowchart of the metered ingress filtering

(b) Flowchart of the gated ingress filtering

Fig. 3: Flowcharts of the proposed filtering approaches

strategy, either threshold-enforcing or blocking, as shown in Fig. 3a. In the threshold-enforcing strategy, a frame is directly dropped if marked as red. Otherwise, if the frame is marked as green, we forward them to the egress to be forwarded to the next hop. In the blocking strategy, red-marked frames are cloned to the CPU port, and the control plane is notified. Then, the control plane removes the registration of this stream, and the frame will be dropped. The green marked frames are handled similarly as in threshold-enforcing.

C. Gated Ingress Filtering

Another data type that the P4 language supports are registers which serve a general purpose and ease the implementation of fully customized algorithms. However, unlike the meters, it is not possible to use registers as per-table-entry or per-stream. Thus, we use a concept called *gates*, which works like a per-class filtering approach in the IEEE 802.1Qci standard. A gate merges the traffic characteristics of multiple streams and handles them as a single stream. Using more gates will allow more fine-grained filtering while increasing the memory requirements. Another critical point here is since it does not enforce per-stream policies, as long as the sum of the traffic at that gate is not exceeded, it does not limit the transmission of any stream. In other words, the allowed transmission capacity for that gate may not be shared equally along the streams assigned to the same gate. However, that is still reasonable as we deal with class-based queuing delays in the TSN.

The basic flowchart of the gated ingress filtering approach is shown in Fig. 3b. When a frame arrives at the switch, a static check is performed as described previously. Then, it looks for a predetermined gate for the stream. If there is no such configuration, the frame will be dropped directly. Otherwise, it fetches the other gate configuration parameters using its gate identifier, GateId.

A core pillar of the gate mechanism is the gate selection function, as it decides which stream is handled by which gate and directly affects filtering. In order to design a *good* gate selection function, there are some issues to be addressed. For instance, deterministic algorithms ease the prediction of the GateId; an attacker with sufficient knowledge can abuse this mechanism to target specific streams by injecting traffic to that gate. Thus, a hash function like a gate selection function would not be appropriate. Another problem can be using large frames as the frame size is checked independently from the gate so that an attacker sending a low number of frames with large frame sizes can inject his/her frames in a gate with streams that have the opposite characteristic and exceed bandwidth by a high degree. Forcing all streams to adhere to the same maximum frame size is a solution but one unsuitable for practical use. A non-deterministic gate selection algorithm is expected to sufficiently mitigate this attack angle, as it does not allow the specific targeting of a gate with a large number of allowed frames.

An important design principle is finding a proper gate selection function to address all mentioned drawbacks. For simplicity, we left the gate selection function selection out of this paper's scope and used a simple strategy: fill empty first (FEF). Thus, it fills empty gates first so that any stream violation will have a limited effect on others as it is also limited to a particular gate. Such an approach may suffer if many streams exist on the gate or the attacker advertises early.

After executing the related gate, a frame counter which is increased for every frame arrival is checked. Here the *Max* value needs to be calculated and configured by the controller. Since different traffic classes in TSN send traffic at varying intervals and a varying number of frames, we used a common observation interval and recalculated the number of frames for that common observation interval. Therefore, this counter checkpoint behaves as a bandwidth check for the gate, which rejects and drops the frame if it exceeds the predefined bandwidth. Unlike metered ingress filtering, the blocking mechanism involves no control plane; gates can be closed at a line rate. This also means less memory as it does not require CPU cloning.

D. Compatibility with the IEEE 802.1Qci Standard

The filtering mechanisms presented in this paper align with the IEEE 802.1Qci standard. In the metered ingress filtering with thresholding, all three steps of the PSFP pipeline are handled by a single table. The forwarding table has an attached colored meter. Therefore, it is both a stream filter and a flow meter. In this case, the relationship between stream filters and flow meters is one-to-one, which is not required, but explicitly allowed in PSFP. The stream gate does not have a technical expression, but conceptually it can be interpreted as always in the *open* state. The metered ingress filtering with a blocking mechanism sends a notification message to the controller once the colored meter is triggered. Then, the associated table entry from the forwarding table is deleted. The PSFP pipeline can conceptualize this by the control plane setting the stream gate to the *closed* state.

In the gated ingress filtering approach, the stream gates are always in the *open* state. Otherwise, the approach is aligned with the PSFP structure. As described in Section III-C, there are two functions, find-gate and execute-gate, which are implemented as tables and are precisely aligned in both the conceptual and the technical sense to the stream filter and the flow meter, respectively. The arbitrary algorithm the flow meter executes is based on a counter reset with each measurement interval and differs between the threshold-enforcing and blocking variants.

IV. Evaluation

In this section, we measure the performance of presented filtering approaches for time-sensitive networks and compare them against the normal case when no filtering solution is applied. For that, we briefly explain the evaluation setup and then evaluate the performance of filtering approaches regarding frame loss and end-to-end latency of frames.

A. Setup

We have implemented the presented dynamic filtering approaches on the P4 behavioral model version (bmv2) and emulated the attacks on Mininet. Note that the employed software switch bmv2 was not designed for performance evaluations and is not necessarily representative of the performance of the mechanisms on a different target. For example, table access could take significantly longer (or shorter) time in a hardware switch. This could significantly influence the results; however, the relative results will remain the same. Apart from emulating the packet processing logic of a P4-Switch on bmv2, we have implemented the control plane in Python 3.6.9. It further interacts with the data plane using the P4Runtime API. All experiments were run on a dual-core Intel(R) Core(TM) i5-7200U CPU with 2.50GHz and 8GB of DDR4 RAM. The machine runs Ubuntu 18.04.6.

As topology, we use a ring topology containing four switches as ring topologies are commonly used in embedded networks, such as cars and factories; it is also common in TSN [20]. As attack traffic, we use a simple attack model known as a babbling idiot in the literature [21], [22]. In the TSN context, a babbling idiot is a talker who correctly advertises traffic and receives a corresponding listener-ready message but then sends more traffic than advertised and exceeds the allocated bandwidth. We randomly placed talkers, listeners, and babblers in the network. As a TSN traffic, we generated 20 random traffic scenarios, including isochronous, cyclic, event-triggered, and best-effort traffic, which are typical TSN traffic classes as described in [8] as follows:

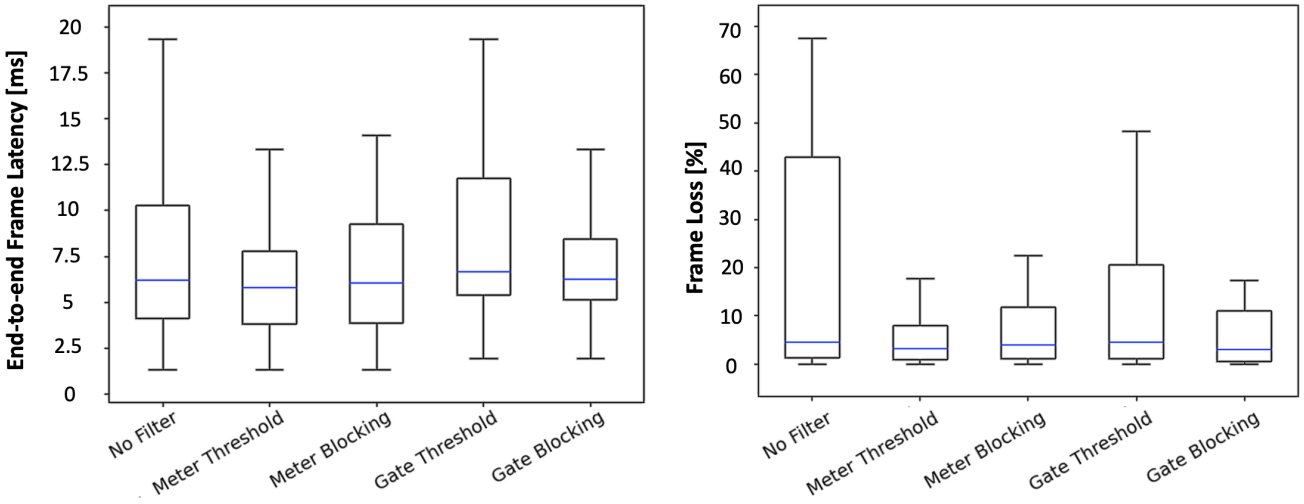
- **Isochronous Traffic:** It is a periodic traffic that requires reserving resources before its period ends. Thus, it can be characterized by an interval and a number of frames per interval. An example of isochronous traffic could be a distance sensor in a car, whose values are constantly required by the emergency brake assistant.
- **Cyclic Traffic:** It is also as periodic as isochronous traffic but contains a fixed length of idle times in between. It sends n frames in every x seconds at an isochronous rate r. An example of cyclic traffic could be a timed sensor or device measurement report.
- **Event Traffic:** It generates single frames sporadically at non-predictable and non-uniform intervals. Network control messages or user input can be an example of such traffic.
- **Best-Effort Traffic:** Most time-sensitive networks also allow a portion of best-effort traffic, i.e., traffic for which the network makes no guarantees regarding arrival or maximum latency. It is sporadic traffic that can be modeled as a random burst.

B. Results

This section evaluates the performance of the presented filtering approaches, Meter, and Gate configured with either Thresholding or Blocking. Then, these approaches are compared with the case there is no filtering applied in terms of their frame loss rate and end-to-end frame latency.

Filtering performance on the delivery of TSN traffic: In order to compare the filtering performance of the presented strategies, we measure the frame loss rate and end-to-end frame latency, which we aim to minimize for legitimate traffic with ingress filtering. Since the results would be highly affected by the type of traffic, we generate 20 scenarios to make a fair comparison between approaches. For that, we generate 60% isochronous traffic with a period of 10 ms, 20% cyclic traffic with a 1-5 s period, 15% event-traffic with a period between 200 ms-1 s, and finally, 5% best-effort-traffic. Results are shown in Fig. 4.

As shown in Fig. 4a, even though the average latencies do not differ significantly between the tested approaches, meter-based filtering, thresholding, or blocking has lower and more bounded end-to-end latencies, which has essential significance in TSN. Also, it might not be sufficient to look only at the latency values as it shows only the end-to-end latency of successfully transmitted frames. Due to the babblers, we see a high number of frame losses in Fig. 4b. It would be misleading to look only at the averages as the TSN promises a certain



(a) End-to-end latency for mixed traffic scenarios

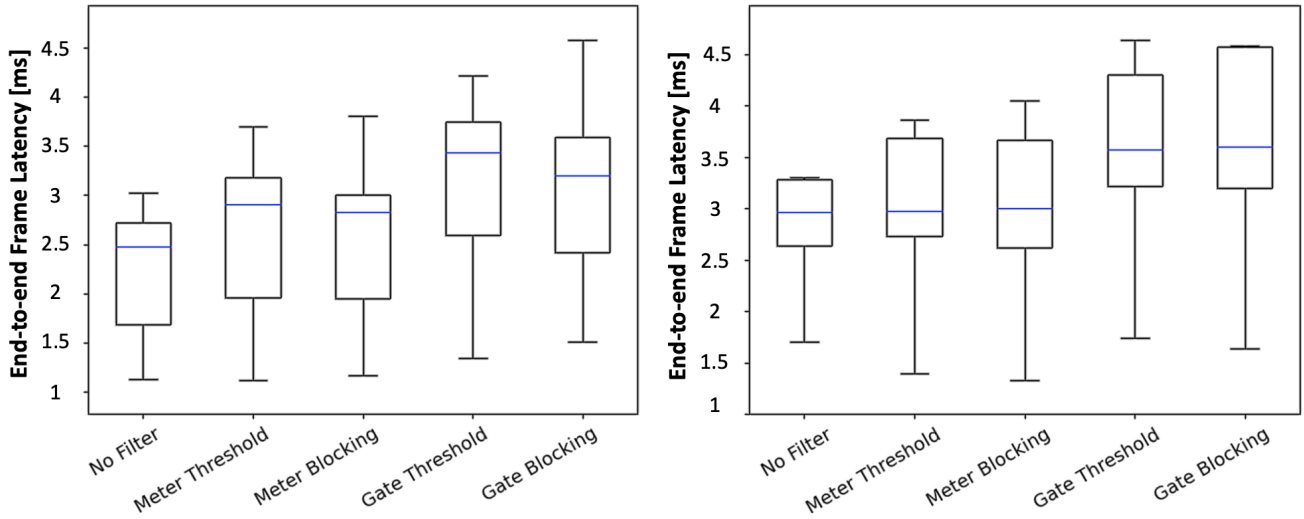
(b) Frame loss for mixed traffic scenarios

Fig. 4: Filtering performance on the delivery of TSN traffic

QoS; the worst-case frame loss rate must also be considered. When *No Filter* is applied, the babblers will significantly affect legitimate streams, and they may experience frame losses up to 68%. It is important to note that the maximum frame loss rate in any filtering approach is well below this value. Here, even in the worst case, we could still say that filtering approaches decrease the frame loss rate of the legitimate streams to ≈24%(as in the upper bound of meter-blocking) and ≈48% (as in the upper bound of gate-thresholding). Thus, it is clear that for some scenarios, they have a huge benefit.

Filtering performance regarding an increasing number of babblers: To analyze the filtering performance in case of an increasing number of bubblers, we simulate single and multi-babblers scenarios and measure how it affects the end-to-end latency of the TSN traffic. For the metered filtering, we set the burst size to 100. In gated filtering, we set the number of gates to 64, and we assumed that if the frame is delayed, we can still tolerate that, and we do not block that stream completely. For that, we set the number of tolerated exceeding frames to half the number of streams currently in the gate. Results are shown in Fig. 5.

Filtering adds additional latency to the frame processing; thus, results in Fig. 5a may be interpreted as the difference between *No Filter* and filtering solutions due to the filtering overhead. However, it should be noted that there is also a babbler in the given test scenario, which further delays the frames of the legitimate stream. To clarify, we repeat the experiment by increasing the number of babblers to three, as



(a) End-to-end latency *with* a babbler.

(b) End-to-end latency *with* 3 babblers.

Fig. 5: Effect of babblers on the end-to-end latency

shown in Fig.5b. The additional latency the multiple babblers caused is minor, around 0.5 milliseconds for the *No Filter* case. For the filtering approaches, the effect of babblers is noticeable but very minor, as we expected. Another result that can be derived from here is the overhead of blocking mechanisms due to cloning to CPU, which also increases as the number of babblers increases. Therefore, they are no longer better than the thresholding mechanisms, as in Fig 5a.

V. Conclusion and Future Work

This paper proposes P4-based dynamic ingress filtering approaches for securing time-sensitive networks from denial-of-service attacks. We proposed a *metered filtering mechanism* that operates per stream and achieves low latency results, even for high traffic demands. Alternatively, we also proposed a *gated filtering mechanism* that fits the per-class filtering concept and enables the deployment of more customizable algorithms. We tested the presented approaches in an emulated mininet environment, and the results show that our filtering approaches can limit frame loss rates of legitimate traffic significantly with only a minimal filtering overhead. Thus, the proposed approaches have the potential to meet strict performance requirements in time-sensitive environments.

As part of our future work, we plan to expand our implementation by incorporating intrusion detection and incident reporting functionality. This enhancement would enable the controller to receive notifications of detected violations, triggering related mitigation mechanisms. Additionally, the PSFP is a proactive approach typically deployed in fixed network positions with a fixed capacity. A promising research direction would be to investigate filtering the attacks that exceed the switch's capacity, for which SDN/NFV-based reactive solutions [23] seem promising as they can flexibly position security functionalities in the network. Moreover, the autonomous configuration of PSFP is another future research direction that aligns perfectly with the concept of self-configured TSN [24]. By adapting related parameters based on changing network conditions, the PSFP's effectiveness could be further enhanced.

References

[1] T. Jeffree, *P802.1Qci – Per-Stream Filtering and Policing*, Sep 2017. [Online]. Available: https://1.ieee802.org/tsn/802-1qci/

[2] S. A. Nsaif and J. M. Rhee, "Seamless ethernet approach," in *2016 IEEE International Conference on Consumer Electronics (ICCE)*, 2016, pp. 385–388.

[3] N. Finn, "Introduction to time-sensitive networking," in *IEEE Communications Standards Magazine*, vol. 2.2, 2018, p. 22–28.

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[5] R. Bifulco and G. Rétvári, "A survey on the programmable data plane: Abstractions, architectures, and open problems," in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018, pp. 1–7.

[6] N. Nayak, U. Ambalavanan, J. M. Thampan, D. Grewe, M. Wagner, S. Schildt, and J. Ott, "Reimagining automotive service-oriented communication: A case study on programmable data planes," *IEEE Vehicular Technology Magazine*, pp. 2–12, 2023.

[7] *Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks - Amendment: 9: Stream Reservation Protocol (SRP)*, 2010. [Online]. Available: https://www.ieee802.org/1/pages/802.1at.html

[8] "Time sensitive networks for flexible manufacturing testbed characterization and mapping of converged traffic types," Mar 2019. [Online]. Available: https://hub.iiconsortium.org/portal/Whitepapers/5eb04d87d2df3f001102b6fe

[9] F. Luo, B. Wang, Z. Fang, Z. Yang, and Y. Jiang, "Security Analysis of the TSN Backbone Architecture and Anomaly Detection System Design Based on IEEE 802.1 Qci," *Security and Communication Networks*, 2021.

[10] 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," *Proceedings of the 6th International OMNeT++ Community Summit*, 2019.

[11] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[12] R. Kandoi and M. Antikainen, "Denial-of-service attacks in openflow sdn networks," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 1322–1326.

[13] D. Merling, W. Braun, and M. Menth, "Efficient data plane protection for sdn," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 10–18.

[14] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, and K.-K. R. Choo, "P4-to-blockchain: A secure blockchain-enabled packet parser for software-defined networking," *Computers & Security*, vol. 88, p. 101629, 2020.

[15] Y. Gao and Z. Wang, "A review of p4 programmable data planes for network security," *Mobile Information Systems*, vol. 2021, pp. 1–24, 2021.

[16] K. Gökarslan, Y. S. Sandal, and T. Tugcu, "Towards a URLLC-Aware Programmable Data Path with P4 for Industrial 5G Networks," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2021, pp. 1–6.

[17] M. Mönnich, N. S. Bülbül, D. Ergenç, and M. Fischer, "Mitigation of IPv6 Router Spoofing Attacks with P4," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '21. New York, NY, USA: Association for Computing Machinery, 2022, p. 144–150.

[18] G. Simsek, H. Bostan, A. K. Sarica, E. Sarikaya, A. Keles, P. Angin, H. Alemdar, and E. Onur, "DroPPPP: A P4 Approach to Mitigating DoS Attacks in SDN," in *Information Security Applications*, I. You, Ed. Cham: Springer International Publishing, 2020, pp. 55–66.

[19] D. J. Heinanen and D. R. Guerin, "A Two Rate Three Color Marker," RFC 2698, Sep. 1999. [Online]. Available: https://rfc-editor.org/rfc/rfc2698.txt

[20] D. Hellmanns, A. Glavackij, J. Falk, R. Hummen, S. Kehrer, and F. Dürr, "Scaling tsn scheduling for factory automation networks," in *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*, 2020, pp. 1–8.

[21] G. Buja, A. Zuccollo, and J. Pimentel, "Overcoming babbling-idiot failures in the FlexCAN architecture: a simple bus-guardian," in *2005 IEEE Conference on Emerging Technologies and Factory Automation*, vol. 2, 2005, pp. 8 pp.–468.

[22] O. Daniel and O. Roman, "Fault injection framework for assessing fault containment of ttethernet against babbling idiot failures," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1–6.

[23] N. S. Bülbül and M. Fischer, "SDN/NFV-based DDoS Mitigation via Pushback," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[24] 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.

Appendix E

Paper 5: Preemptive DoS attacks on Time Sensitive Networks

Abstract

Time-sensitive networking (TSN) is a promising technology for real-time communication in industrial and auto- motive networks. The frame preemption is one of its key features, which allows high-priority traffic to interrupt the transmission of low-priority traffic to decrease the delay of critical traffic. However, the deterministic nature of TSN frame preemption also renders it vulnerable to denial of service (DoS) attacks that can significantly reduce the quality of service (QoS) of flows by increasing delays and packet loss. This paper introduces the concept of preemptive DoS attacks and evaluates their impact on TSN QoS performance. For that, we describe a strategy that attackers can use to estimate the used preemption scheme in the switch and then show how an active attacker can use this information to degrade the QoS of TSN. Our simulation results indicate that even a single attacker can significantly deteriorate the QoS of TSN traffic. To counter such attacks, we also discuss possible countermeasures, such as dynamic changes in the runtime to limit the knowledge of the attacker in this paper.

Reference

Nurefşan Sertbaş Bülbül and M. Fischer. Preemptive DoS attacks on Time Sensitive Networks. Accepted and to appear at IEEE Global Communications Conference (GLOBECOM), 2023. ©2023 IEEE.

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.

Preemptive DoS attacks on Time Sensitive Networks

Nurefşan Sertbaş Bülbül and Mathias Fischer
Department of Informatics, Universität Hamburg, Germany
Email:{nurefsan.sertbas, mathias.fischer}@uni-hamburg.de

***Abstract*—Time-sensitive networking (TSN) is a promising technology for real-time communication in industrial and automotive networks. One of its key features is frame preemption, which allows high-priority traffic to interrupt the transmission of low-priority traffic, thereby reducing the delay of high-priority critical traffic. However, the deterministic nature of TSN frame preemption also makes it vulnerable to denial of service (DoS) attacks, which can severely impact flow quality of service (QoS) by increasing delays and packet loss. In this paper, we introduce the concept of preemptive DoS attacks and evaluate their impact on TSN QoS performance. We describe a strategy that attackers can use to estimate the preemption scheme configured in the switch and then demonstrate how an active attacker can use this information to degrade TSN QoS. Our simulation results indicate that even a single attacker can significantly deteriorate the QoS of TSN traffic. It is important to address this vulnerability in TSN and develop countermeasures to prevent preemptive DoS attacks from occurring.**

***Index Terms*—TSN, frame preemption, time-sensitive networking, traffic analyzing attacks, calibrated attacks, denial of service**

I. Introduction

Real-time communication has become increasingly critical in various industries, such as avionics, industrial, and automotive, where rapid and reliable data exchange is essential for safe and efficient operation. In these industries, traffic flows generated by time-critical (TC) applications with stringent requirements for bounded latency and low jitter often share communication channels with the flows originating from non-time-critical (NTC) applications. However, standard Ethernet does not provide any inherent guarantees for delivering time-sensitive data as it was not initially conceived for real-time applications and strict timing requirements.

In standard Ethernet, frames transmitted non-preemptively may experience significant queuing delays if there is no mechanism to interrupt transmissions. In the worst case, a high-priority TC frame may be delayed by a lower-priority NTC frame already being transmitted. Once the NTC frame has completed transmission, the TC frame will be processed as soon as possible. This may be fine for light traffic scenarios. However, for heavy traffic, long queuing delays might be the result. Using jumbo frames whose maximum transmission unit could be larger than 9k bytes can cause delays around 120 μs per 100 Mbps to switch in the worst case [1], [2]. This is a considerable latency for TC traffic.

The IEEE Time-Sensitive Networking task group has proposed standards that enhance Ethernet with real-time capabilities and various QoS classes to address that problem. Particularly, the IEEE 802.3br and IEEE 802.1Qbu standards propose a frame preemption mechanism allowing the preemption of certain traffic classes in favor of other classes [3], [4]. With that, frames configured as express can suspend transmission of the frames configured as preemptable. Hence, it enables processing and forwarding high-priority frames immediately upon arrival without being delayed by low-priority frames. Furthermore, it has been shown that with preemption, the waiting time for higher-priority packets becomes independent of the number of low-priority packets [5].

Although TSN frame preemption is crucial to TSN's real-time capability, it also creates potential security risks. It still suffers from worst-case delays that could be orders of magnitude longer than average, even when using only a single express traffic class [6]. Moreover, it can cause the starvation of NTC frames and exhaustion of full switch buffers [1]. These cases can occur spontaneously in a regular network and can be abused by attackers to target specific flows or, more generally, to increase the queueing delay. For instance, an attacker could send express traffic to the target switch just before the targeted traffic, causing long waiting times and a DoS. Therefore, the preemption mechanism requires further research, as it is a relatively new addition to the TSN family, and its vulnerability to preemptive DoS attacks needs to be thoroughly investigated.

This paper explores the vulnerabilities of TSN frame preemption and their implications for TSN networks. For that, an attacker first needs to passively monitor the network to extract useful information, such as the used configuration scheme in the preemption mechanism. Then, based on the observations, (s)he can attack the preemption mechanism. Accordingly, our contributions are:

- We identify various potential attack scenarios and discuss their possible impacts on network performance. In the process, we introduce the concept of preemptive DoS attacks.
- We demonstrate how even the passive observation of a TSN, e.g., monitoring a single input and output port of a switch, can be prolonged by attackers. Furthermore, with the OMNeT++ simulations, we show how feasible using the collected information to launch attacks is. Our simulation results indicate that even a single attacker can cause significant delays for specific traffic classes.

The rest of this paper is structured as follows: Section II briefly describes the frame preemption mechanism and describes the state of the art. In Section III, we examine the vulnerabilities of the preemption mechanism and define

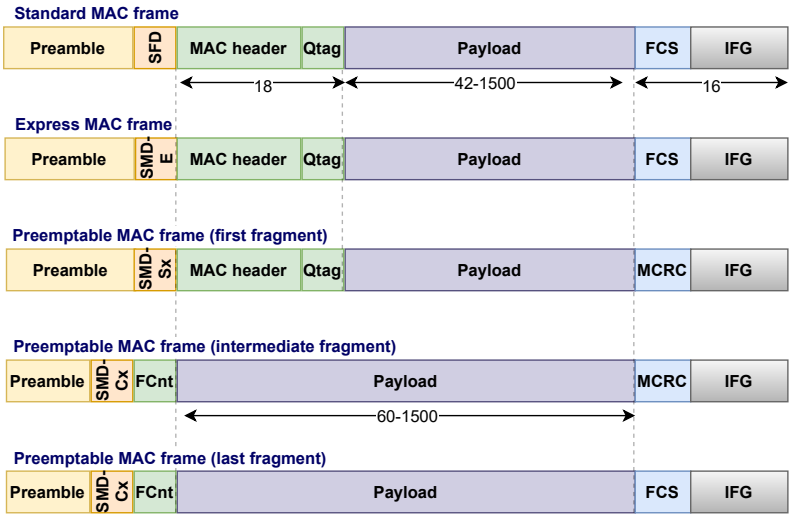


Fig. 1: MAC Frame Format

threats. We give feasibility results in Section IV. Finally, Section V concludes the paper and summarizes future work.

II. Background and Related Work

Frame preemption is a TSN mechanism that allows critical traffic to interrupt the lower criticality traffic transmission to satisfy strict timing requirements. Since preemption is defined on a transmission port of a switch, it requires changes both in the medium access control (MAC) layer and bridge management protocol. Accordingly, these two mechanisms are described in two standards: IEEE 802.1Qbu for bridge management and IEEE 802.3br for the Ethernet MAC.

At the egress port of the bridge, frames are handled differently depending on the configuration. The express MAC (eMAC) and preemptable MAC (pMAC) layers handle frames differently depending on whether they are configured as express or preemptable. Preemption can only occur between express and preemptable frames, and frames in the same class cannot preempt each other. Besides, frames of express classes cannot be preempted; they can only preempt preemptable frames. In preemption, the preempted frame's transmission is resumed until the express frame's transmission finishes. Therefore, it is ensured that high-priority traffic configured as express can no longer be delayed by low-priority preemptable traffic. The preempted frame can continue transmission from where it left off only after this point.

As a result of the preemption, preemptable frames are split into fragments. To make the frame preemption transparent to the Ethernet's physical layer, the basic MAC frame structure is preserved as shown in Fig. 1. Each frame starts with a preamble and ends with a cyclic redundancy check, FCS or MCRC, and an inter-frame gap (IFG). In express frame format, the start frame delimiter (SFD) field is replayed by SMD-E (start MFrame delimiter - express) to signal the express frame afterward. The first fragment of the preemptable frame also has a very similar structure, signaling by SMD-Sx (SMD - start fragment) field. Following the first one, all fragments are signaled by SMD-Cx (SMD - continuation fragment) and a fragment counter (FCnt). This way, fragments can be combined at the MAC layer in order. Since IEEE 802.3br only allows one level of preemption, the MAC header and Q-Tag fields are transmitted once in the first fragment, allowing a bigger payload in the later fragments.

According to the standards, all MAC frames/fragments must meet the minimum Ethernet frame size requirement of 84 bytes. For the smaller fragments, the payload needs to be padded accordingly. However, padding is not allowed in the preemption, which results in some constraints to the payload sizes in the fragments. The smallest frame that can be preempted has a payload of 102 bytes, as it can be split into 42 bytes and 60 bytes to form a start and continuation fragment that fulfills the minimum Ethernet frame size. Considering the frame formats, the longest lower priority frame/fragment that can block an express frame is 143 bytes long, as proven in [2].

There have been several studies on the worst-case analysis of frame preemption and its effects on end-to-end transmission delays, as described in [1]. The authors tested a custom preemption mechanism in [7], and the implementation results confirm that latency and jitter of real-time traffic are reduced compared to standard priority-based Ethernet in the mixed traffic scenarios. For the practical use of frame preemption, an optimization framework was presented in [8] for assigning frames to preemptable or express queues and then deciding the optimal allocation of the queues as preemptable or express. The results of this study showed that the queuing configuration had a considerable impact on performance.

In [9], authors state a problem that an express frame can cause long delays over the preemptable frames. Also, since preemptable frames cannot preempt each other in some

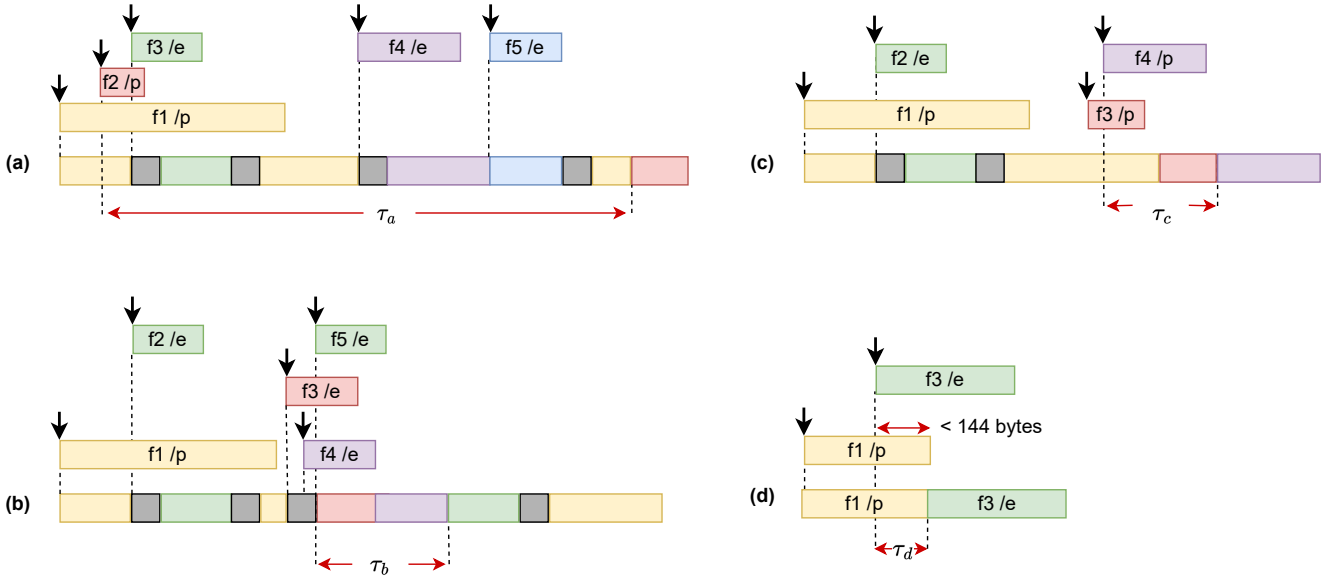


Fig. 2: Preemption effect on particular scenarios: (a) express frames to block preemptable frames, (b) express frames delaying each other, (c) preemptable frames delaying each other, (d) preemptable frames to block express frames

scenarios, huge delays may occur regardless of the timing constraints of preemptable frames. Thus, the authors propose a modification of the MAC layer by adding a third interface named tpMAC. So, express frames can preempt all classes, and frames assigned to tpMAC can also preempt preemptable frames. This way, the waiting time for the preemptable traffic classes with timing constraints can be shortened. The implementation aspects of this preemption mechanism are discussed in [10]. The authors focus mostly on implementation details and present a scalable architecture that can run in application-specific integrated circuits (ASIC) and field-programmable gate array (FPGA) platforms. Another solution to that problem is proposed in [11]. It requires modification of the pMAC and does not add a third interface as in [9]. Simulation results confirm that the presented mechanism achieves better response times for high-priority frames and only slightly impacts the response times of low-priority frames.

Recent work in [11] addresses the limitations of the current preemption mechanism. They claim that high-priority frames can experience significant blocking delays in certain cases, and they propose a credit-based preemption mechanism to lower the maximum blocking time of the express frames. However, to our knowledge, the preemption mechanism's vulnerabilities have yet to be discussed from the attacker's perspective in related work. This paper analyzes the limitations and malicious use of the standard frame preemption mechanism by simulations in a realistic TSN environment. Then, a few candidate solutions will be discussed to handle these limitations.

III. TSN Frame Preemption Scenarios

This section analyzes the preemption mechanism and lists its performance shortcomings for low and high-priority frames.

A. Preemption Effect on Particular Service Types

Frame preemption is already proposed to handle possible misuse of low-priority frames for targeting timing constraints of the high-priority frames. For that, high-priority express traffic is saved from the long waiting times due to low-priority traffic. However, even though it helps high-priority traffic preserve its timing constraints, frame preemption still has shortcomings. In the following, we list the effects of the different preemption cases on certain traffic classes.

Scenario-a: Express frames block preemptable frames: The frame preemption mechanism can decrease significant waiting times of high-priority traffic. However, a preemptable frame can be preempted multiple times, which induces overhead as a part of the frame-splitting process that can significantly delay the transmission of preemptable traffic. This may also lead to the starvation of preemptable frames and the exhaustion of switch buffers. As can be seen in Fig. 2-a, even though the preemptable frame f_1, arrives before the express frames, f_3, f_4, f_5, it has to wait. As in this example, sending consecutive express frames results in multiple preemptions of f_1 and, therefore, long delays. Also, the other preemptable frame, f_2, faces a similar long waiting time.

Scenario-b: Express frames delay each other: Even though express frames can not preempt each other, they can still cause delays. An attacker can send certain priority frames before the reception of a time-critical high-priority frame, as shown in Fig. 2-b. Suppose that green frames are periodic and the attacker has extracted that period. The attacker can insert higher priority frames before the next transmission of f_2 (so f_5). In that case, f_5 needs to wait for the transmissions of f_3 and f_4. The attacker can adjust the number of inserted frames considering the extracted transmission period to delay

the transmission of f_5 and violate its timing constraints. Furthermore, in the worst case, all express streams in the network use the same egress port.

Scenario-c: Preemptable frames delay each other: According to the standard preemption mechanism, preemptable frames cannot preempt each other. If preemptable frames are forwarded in a first come, first served manner, as suggested in the standards, some low-priority but critical frames may suffer from long waiting times. Such frames are classified as preemptable but still may have softer timing constraints [12]. However, attackers can send a long preemptable frame before transmission to delay such lower-priority frames for longer periods. As can be seen in Fig. 2-c, if another preemptable frame f_3, would be inserted just before the time critical but low priority preemptable frame f_4. In this case, f_4 needs to wait until f_1 and f_3 have been transmitted.

Scenario-d: Preemptable frames block express frames: Even though neither express nor preemptable frames can preempt express frames, there are some exceptions in theory. As we mentioned in Section II, further preemption is not allowed if the fragment size does not meet the minimum Ethernet frame size requirement. Thus, the express frame needs to wait for the transmission of 143 bytes which is the longest non-preemptable size in the worst case, as can be seen in Fig. 2-d.

B. Configuration of Preemption Service Classes

Apart from the blocking possibilities of certain traffic services to each other, an improper configuration of these classes can also lead to threats. A higher-priority queue can be mapped to a pMAC, and a lower-priority queue can be mapped to an eMAC. Even though we do not expect such scenarios in practical implementations, improper or illegitimate configurations may occur.

IV. Evaluation

In this section, we simulate a simple TSN network to show vulnerabilities of the standard frame preemption mechanism considering the defined attack scenarios presented in Section III. First, we briefly explain the evaluation setup and the attacker model. Then, we show the feasibility of the attacks on the TSN preemption mechanism. Finally, we summarize our simulation results.

A. Experimental Setup

We use OMNeT++ discrete event simulation framework for the simulative analysis with NeSTiNg simulation model developed for time-sensitive networks by extending inet library [13], [14]. For our experiments, we used a simple topology as shown in Fig. 3 in which we have two legitimate transmitting end-hosts per class, a *talker* in TSN, and one malicious *talker*, a single TSN switch and single receiving end-hosts, *listener* in TSN, as we only need to monitor the ingress and egress of a single switch. Also, we set the link capacity to 100 Mbps for simplicity. Finally, the experiments are repeated for 20 scenarios, and results are given with confidence intervals.

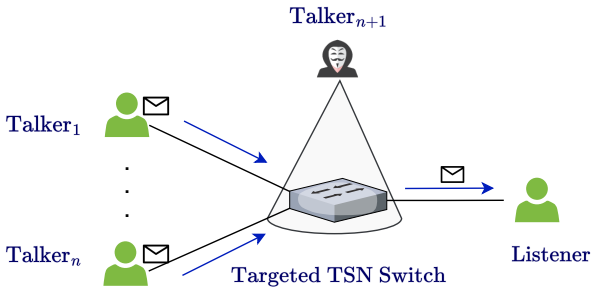


Fig. 3: Test topology

TABLE I: Simulated TSN traffic

	Type	Period	Deadline	Payload size
Q7	Control	10-40 ms	Same as period	10-100 Byte
Q6	Sensor	40-100 ms		100-200 Byte
Q5	CAM, Radar	20-100 ms		400-1500 Byte
Q0	Best effort	50-100 ms	-	1500 Byte

Since there is no publicly available data set for TSN traffic, we obtained the traffic generation parameters from TSN literature and tried to model TSN traffic as realistically as possible [8]. Considering in-vehicle networks as a use case, we generated four traffic classes; control signal, sensor data, raw data from camera or radar (lidar), and best-effort traffic, as shown in Table I. The best effort in this scenario does not require any timing guarantee and represents the non-time critical traffic. On the other hand, the remaining traffic classes are usually used by time-critical applications; thus, these data flows have specific traffic deadlines and represent time-critical traffic.

We also assume the preemption mechanism is appropriately configured as described in Section III-B. Hence, we assume low-priority queues are not configured as express when there is high-priority traffic in the network. We listed a few configuration scenarios in Table II. Here, *P* represents a related queue configured as preemptable, while *E* represents an express queue. For instance, in the third configuration, queues seven (Q7) and six (Q6) are configured as express, while others are configured as preemptable.

TABLE II: Possible frame preemption configurations

Configuration	Q7	Q6	Q5	Q0
#1	P	P	P	P
#2	**E**	P	P	P
#3	**E**	**E**	P	P
#4	**E**	**E**	**E**	P
#5	**E**	**E**	**E**	**E**

B. Evaluation Metrics

We used the following metrics to evaluate our scenarios:

- Observed Mean Variance in Delay: The mean-variance of the observed delay of either the applied scenario (e.g., class i traffic injection) or the initial status where no traffic is injected.

- Mean Stream Latency (MSL): The average latency of streams until they reach their destination.

C. Attacker Model

To show how the malicious talkers can abuse the preemption mechanism, the attack model needs to be defined, stating its capabilities. Here, two different attack models can be used as follows:

- *Attackers with full knowledge:* Attacker knows everything about the network. For instance, the attacker can calculate upper bounds for the queuing delay at each hop as (s)he knows about talkers' behavior.
- *Attackers with partial knowledge:* Here, the attacker can either (i) passively monitor the network and capture traffic or (ii) actively send certain priority probe packets to the network. Then, the attacker analyzes the time deltas between sent messages and their corresponding responses on specific links. Then, the attacker may perform traffic-analysis attacks to learn when a particular node forwards a specific frame or where it will be processed at the given time.

An attacker with full knowledge can manipulate the network easily due to the deterministic nature of TSN. However, it becomes challenging with limited knowledge. Thus, this paper assumes that the attacker initially has zero knowledge. Furthermore, we only assume that the switch configuration does not change during our observation period, which we select as $100\mu s$. During this period, the attacker monitors a series of messages at the ingress and egress of the target switch and sees the switch as a black box. Furthermore, the attacker can control a few talkers. In that case, it can inject traffic from a specific class to delay certain traffic flows.

D. Results

This section presents how an attacker can perform traffic analysis and how the scenarios defined in Section III become feasible based on the attacker's observations.

Traffic analysis: Assuming that the switch is a black box for an attacker, we only monitor the ingress and egress of the TSN switch and hold statistics per class, such as experienced frame delay. Thus, the attacker can perform traffic analysis by analyzing the relations between packets passing through the observed switch. For that, we record the arrival and departure times of the frames for all classes and extract minimum, maximum, and average latency and the variance of the latency values per class from the observed latency values. Preliminary results show that minimum, maximum, and average latency values differ significantly based on the generated traffic scenarios. However, relating these metrics with the used frame preemption configuration is hard.

Thus, we use the variance of the measured latency by injecting different classes of traffic and see how much it affects the latency of other classes (See Fig. 4). However, that does not require active traffic injection of the attacker; it may also be possible that the attacker just passively monitors the network where new talkers join the network and start sending

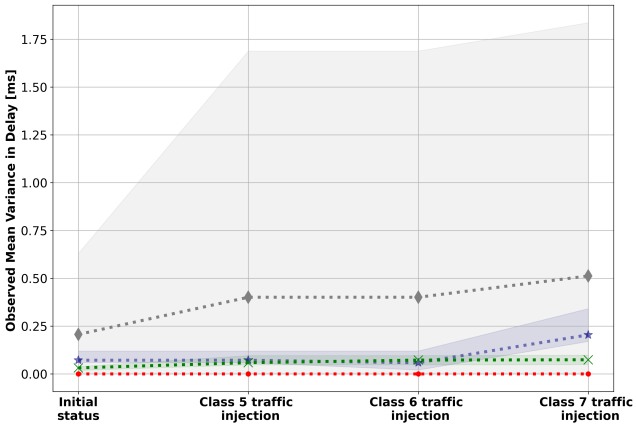


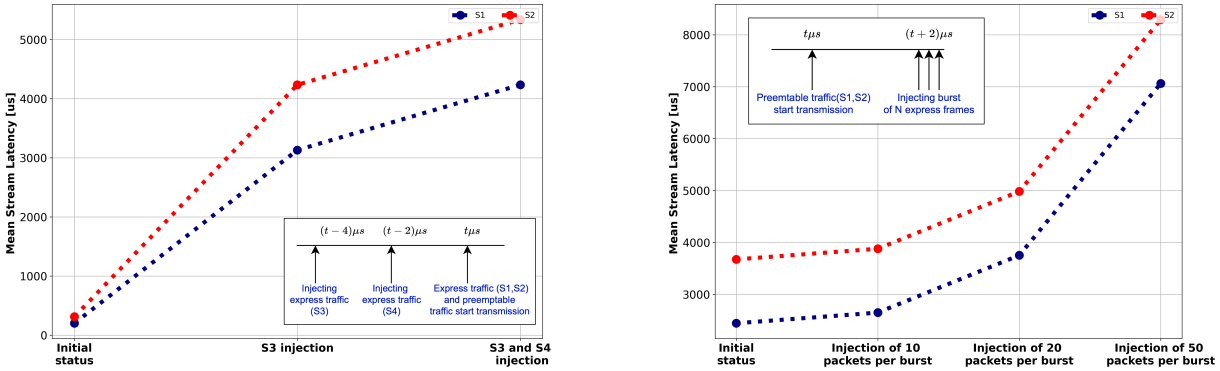
Fig. 4: Attacker observations in case of configuration 2 is used

traffic. Results of configuration #2, where only Q7 is set as an express, are shown in Fig. 4. It can be differentiated from the other classes independent of the injected traffic class; it is unaffected as it has a right to preempt other classes. In some cases where two express frames arrive simultaneously, we may see slight delays. However, this effect does not vary the results. Moreover, experiments using the other configurations in Table II resulted in parallel results and confirmed our claim that variance is an excellent metric for estimating the used configuration scheme, e.g., express traffic classes.

Injecting malicious traffic into the network: After the traffic analysis, the attacker can predict a specific traffic class assigned to a preemptable or an express. Then, using this information, it can perform the scenarios described in Section III. To show its feasibility, we simulate two scenarios and measure the success of the attacks in terms of the mean latency of the streams. Finally, we configure the Q7 as an express and the remaining queues as preemptable. For simplicity, we set the link capacity to 10 Mbps so that it is possible to see the preemption effect even with a small scenario.

In the first scenario, we generate two express streams, S1 and S2, and two preemptable streams. Then, we measure the mean end-to-end latency of express traffic as shown in Fig. 5a. Here, we left minimum and maximum values out as they are very narrow. Initially, we see that S1 and S2 have very low MSL, around 250 μs. Then, assuming the attacker passively monitored the network and detected S1 and S2 as express traffic, s(he) sends express traffic before S1 and S2 start transmission. Here, we set this value as $2\mu s$ so that, as shown in the timing diagram, $2\mu s$ before the transmission of S1 and S2, the attacker injects express traffic. Since express frames cannot preempt each other, we see a significant increase when malicious express traffic, S3, and S4, is injected into the network. The preliminary results show that it is possible to carry out a calibrated attack to delay the targeted streams further. In the worst case, this may cause a violation of QoS constraints, e.g., missing deadlines.

In the second scenario, to show how preemptable traffic is affected by the express traffic, we initially generate two



(a) Injecting express traffic to delay specific express stream(s)

(b) Injecting express traffic to delay specific preemptable stream(s)

Fig. 5: Injecting malicious traffic into the network

preemptable streams, S1 and S2. Then, we measure the mean end-to-end latency of preemptable traffic as shown in Fig. 5b. In case the attacker sends express frames during the transmission of preemptable frames, they can be preempted. For that, we inject express frames as bursts. Results claim that even a single express stream that generates multiple bursts periodically can cause a significant delay of preemptable traffic. In the worst case, preempting several times may cause the starvation of the preemptable traffic.

V. Conclusion

Frame preemption is one of the mechanisms in TSN that helps limit the waiting time of certain traffic classes so that strict QoS requirements can be satisfied. However, security has yet to be one of the main design concerns. Since the primary application domains of the TSN protocols are mission-critical systems, it is crucial to explore potential threats and apply candidate countermeasures. This paper presents a few vulnerabilities of the standard frame preemption mechanism and describes scenarios that malicious end hosts can abuse. Then, we simulate a typical TSN network and show how much useful information an attacker can extract and how this information can potentially be used to violate the QoS constraints of TSN traffic so denial of service.

In our future work, we intend to explore more intricate scenarios involving multiple switches to enhance the impact of attackers, even though a single switch can already cause significant delay. Additionally, to mitigate these attacks, we intend to deploy different strategies limiting the attacker's knowledge of the network using the moving target defense (MTD) concept. By introducing diversity into the network, MTD can randomize certain features and prevent attackers from predicting network traffic. This will make it harder for attackers to identify patterns and predict future events despite an increased attack budget. Specifically, we aim to make it difficult for attackers to determine when a particular flow is being forwarded by a specific TSN node or where it is being processed at any given time. These efforts are part of our broader goal to enhance the security of time-sensitive networks.

References

[1] W.-K. Jia, G.-H. Liu, and Y.-C. Chen, "Performance evaluation of IEEE 802.1 Qbu: Experimental and simulation results," in *38th Annual IEEE Conference on Local Computer Networks*. IEEE, 2013, pp. 659–662.

[2] D. Thiele and R. Ernst, "Formal worst-case performance analysis of time-sensitive Ethernet with frame preemption," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–9.

[3] "IEEE Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic," *IEEE Std 802.3br-2016 (Amendment to IEEE Std 802.3-2015 as amended by IEEE St802.3bw-2015, IEEE Std 802.3by-2016, IEEE Std 802.3bq-2016, and IEEE Std 802.3bp-2016)*, pp. 1–58, 2016.

[4] "IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption," *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)*, 2016.

[5] Z. Zhou, Y. Yan, S. Ruepp, and M. Berger, "Analysis and implementation of packet preemption for time sensitive networks," in *2017 IEEE 18th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2017, pp. 1–6.

[6] D. Hellmanns, J. Falk, A. Glavackij, R. Hummen, S. Kehrer, and F. Dürr, "On the Performance of Stream-based, Class-based Time-aware Shaping and Frame Preemption in TSN," in *2020 IEEE International Conference on Industrial Technology (ICIT)*, 2020, pp. 298–303.

[7] J. Kim, B. Y. Lee, and J. Park, "Preemptive switched ethernet for real-time process control system," in *2013 11th IEEE International Conference on Industrial Informatics*. IEEE, 2013, pp. 171–176.

[8] T. Park, S. Samii, and K. G. Shin, "Design optimization of frame preemption in real-time switched ethernet," in *2019 Design, Automation and Test in Europe Conference (DATE)*. IEEE, 2019, pp. 420–425.

[9] M. A. Ojewale, P. M. Yomsi, and B. Nikolić, "Multi-Level Preemption in TSN: Feasibility and Requirements Analysis," in *IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*, 2020.

[10] M. Knezic, M. Kovacevic, and Z. Ivanovic, "Implementation Aspects of Multi-Level Frame Preemption in TSN," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1127–1130.

[11] M. Ashjaei, M. Sjödin, and S. Mubeen, "A novel frame preemption model in TSN networks," *Journal of Systems Architecture*, vol. 116, p. 102037, 2021.

[12] M. A. Ojewale, P. M. Yomsi, and B. Nikolić, "Multi-level preemption in TSN: feasibility and requirements analysis," in *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2020, pp. 47–55.

[13] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, and K. Rothermel, "NeSTiNg: Simulating IEEE time-sensitive networking in OMNeT++," in *Proceedings of the 2019 International Conference on Networked Systems*, Garching b. München, Germany, Mar. 2019.

[14] "INET framework." [Online]. Available: https://inet.omnetpp.org/

Appendix F

Paper 6: SDN/NFV-based DDoS Mitigation via Pushback

Abstract

Distributed Denial of Service (DDoS) attacks aim at bringing down or decreasing the availability of services for their legitimate users, by exhausting network or server resources. It is difficult to differentiate attack traffic from legitimate traffic as the attack can come from distributed nodes that additionally might spoof their IP addresses. Traditional DoS mitigation solutions fail to defend all kinds of DoS attacks and huge DoS attacks might exceed the processing capacity of routers and firewalls easily. The advent of Software-defined Networking (SDN) and Network Function Virtualization (NFV) has brought a new perspective for network defense. Key features of such technologies like global network view and flexibly positionable security functionality can be used for mitigating DDoS attacks. In this paper, we propose a collaborative DDoS attack mitigation scheme that uses SDN and NFV. We adopt a machine learning algorithm from related work to derive accurate patterns describing DDoS attacks. Our experimental results indicate that our framework is able to differentiate attack and legitimate traffic with high accuracy and in near-realtime. Furthermore, the derived patterns can be used to create OpenFlow (OF) or Firewall rules that can be pushed back into the direction of the attack origin for more efficient and distributed filtering.

Reference

Nurefşan Sertbaş Bülbül and M. Fischer. SDN/NFV-based DDoS Mitigation via Pushback. IEEE International Conference on Communications (ICC), 2020. ©2020 IEEE.

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.

SDN/NFV-based DDoS Mitigation via Pushback

Nurefşan Sertbaş Bülbül and Mathias Fischer
Department of Computer Science, University of Hamburg, Germany
Email:{sertbas,mfischer}@informatik.uni-hamburg.de

***Abstract*—Distributed Denial of Service (DDoS) attacks aim at bringing down or decreasing the availability of services for their legitimate users, by exhausting network or server resources. It is difficult to differentiate attack traffic from legitimate traffic as the attack can come from distributed nodes that additionally might spoof their IP addresses. Traditional DoS mitigation solutions fail to defend all kinds of DoS attacks and huge DoS attacks might exceed the processing capacity of routers and firewalls easily. The advent of Software-defined Networking (SDN) and Network Function Virtualization (NFV) has brought a new perspective for network defense. Key features of such technologies like global network view and flexibly positionable security functionality can be used for mitigating DDoS attacks. In this paper, we propose a collaborative DDoS attack mitigation scheme that uses SDN and NFV. We adopt a machine learning algorithm from related work to derive accurate patterns describing DDoS attacks. Our experimental results indicate that our framework is able to differentiate attack and legitimate traffic with high accuracy and in near-realtime. Furthermore, the derived patterns can be used to create OpenFlow (OF) or Firewall rules that can be pushed back into the direction of the attack origin for more efficient and distributed filtering.**

***Index Terms*—DDoS, pushback, pattern generation, NFV, SDN**

I. Introduction

DDoS attacks are malicious attempts to interrupt services by congesting networks, consuming server resources, or disrupting the availability of the network components. It is hard to differentiate DoS attack traffic from legitimate traffic, as attackers also try to mimic legitimate traffic and can easily spoof IP source addresses.

Moreover, so-called Flash Crowds (FC) that manifests in large amounts of traffic from legitimate users occur naturally and hard to differentiate from DoS attacks as they share similar traffic characteristics [1]. Only when attack traffic can be clearly differentiated from legitimate traffic, the attack traffic can be filtered without affecting the legitimate traffic. For that, several network traffic analysis methods have been proposed in the literature. Although some of the methods can be easily implemented, they have poor performance and low detection accuracy [2], [3].

Current mitigation solutions are mostly based on dedicated appliances such as firewalls. However, they use predefined security policies that fail to identify and drop new attack patterns. Cloud-based mitigation solutions have resources to handle these attacks, but cannot guarantee end-to-end security. For this reason, not all service providers want to use them.

Recently, the introduction of SDN and NFV technologies facilitates new network security solutions. NFV decouples the software implementation of network functions from the underlying hardware by virtualization, and therefore functions can be started when and where they are needed. SDN can be seen complementary to NFV and can be used to manage NFV infrastructure with the help of SDN specific features like global network view, traffic analysis, and the configuration of dynamic packet forwarding. By leveraging SDN and NFV features, it is possible to defend the network against DDoS attacks even for huge traffic volumes that normally would exceed the deployed firewall capacity [4]–[7].

An efficient mitigation should minimize false positives and maximize true positives so that only malicious traffic can be identified and blocked. Considering the current attacking schemes, the system should be able to deploy on-demand functions when and where they are needed so that the system can react to varying types and sizes of attacks. While doing so, it needs to operate with low latency and react attacks on time.

In this paper, we propose an SDN-NFV-based DDoS attack mitigation framework that filters malicious traffic and pushes back the filtering rules upstream into the direction the malicious traffic is coming from. Benefitting from the idea that carrying out some tasks locally may help to achieve a common goal globally [8], our mechanism detects attack locally and pushes it back. Such a mechanism can efficiently drop attack traffic close to its origin [9], [10]. However, to filter malicious traffic an aggregate of the malicious DoS traffic is needed that can be easily implemented either as SDN flow rule or a firewall rule. For that, we propose to apply the generalization and summarization algorithm, AOI, in this domain [11]. It extracts patterns that describe the majority of the traffic, which usually manifests in a pattern for the DoS traffic during an attack. Afterwards, the attack can be mitigated via SDN/NFV by informing intermediate routers to filter the traffic that fits the pattern. Our paper makes thus the following contributions:

- We revise the original Pushback mechanism introduced in [9] with the use of SDN/NFV to handle the limitations of traditional networks. SDN eases the collaboration not only between the forwarding entities but also between networks. Therefore, attacks can be directly mitigated by OF. We might also need NFV to pushback the attack if we are outside of the SDN domain.
- We evaluated AOI for deriving filtering rules with up-to-date data sets that contain different types of DoS attacks. We conclude that it can effectively differentiate attack from legitimate traffic, and can be used for deriving filtering rules for attack traffic. These rules can be used at upstream routers to filter traffic closer to its origin.

- Our results indicate that the proposed algorithm works independently of the type of DDoS attack. Results show that AOI can identify the attack and legitimate traffic with high accuracy. It drops 99.25% of the attack traffic while dropping only 7.61% of the legitimate traffic on average. When combined with sampling, it allows for a near-realtime derivation of filtering rules.

The remainder of this paper is structured as follows: Section II summarizing current DDoS attack detection and mitigation approaches. Section III presents a novel DDoS mitigation framework using NFV and SDN. We give some simulation results and evaluate our approach in Section IV. Finally, Section V concludes the paper and outlines our future work.

II. Related Work

In this section, we will survey the literature on pattern generation and the mitigation of DoS attacks.

a) Pattern Generation: Patterns may be extracted from the network traffic by analyzing attributes like IP addresses, protocols and port numbers. These patterns are useful for describing how the attack looks like. Therefore, generating a proper pattern significantly affects attack mitigation performance. A generated pattern should be accurate enough otherwise, it may filter unintended packets. Pattern generation plays a significant role in the concept of Aggregate-based Congestion Control (ACC) [9]. The idea behind it is that an overall increase in the traffic cannot be described by a single flow because attack traffic is usually spread across more than one flow. Therefore, traditional flow-based solutions fail to identify an aggregate of packets, which can be defined as a collection of packets coming from one or more flows that share the same attributes, e.g, destination IP addresses [10].

The authors introduce a method called Longest Matching Prefix (LMP) to describe aggregates in the concept of ACC. It is used in routers to choose a forwarding entry from a routing table. In the ACC concept [10], attack patterns are generated based on the longest matching prefix of the dropped traffic by routers. Destination IP addresses of the dropped packets are extracted and sorted by prefixes. The most frequent prefix is selected as a signature of the congestion. Then, this signature is used for filtering the traffic that causes congestion at the corresponding router and upstream routers. That approach operates near to real-time and is easy to implement as it only uses destination IP addresses. However, LMP fails to differentiate legitimate traffic from attack traffic, when not a complete network but a single server is targeted. In such a scenario, the generated signature matches all traffic designated to the server and drops the legitimate traffic as well.

As an alternative to LMP, AOI can be adapted to this domain for generating attack patterns. AOI is a summarization machine learning algorithm that learns from example techniques [12]. The main difference between machine learning is derived from the method of generalization. While data is generalized on tuple by tuple basis in machine learning, AOI uses attribute by attribute generalization. It is an effective approach to extract rules from a set of data and generate high-level representations. Thus, it has been used in different fields like alarm clustering and, summarizing the relational databases. In [13], it is used for clustering thousands of alarms into clusters. The algorithm repeatedly replaces alarm attributes with more generalized tags which are defined by generalization hierarchies. At the end, alarms are represented more compact and has more compact representations. In other words, general information about the alarm, e.g., a privileged or a non-privileged port was used, is kept instead of its actual values e.g., a specific port number. By merging similar alarms, clusters are generated and the biggest cluster produced by AOI reflects the majority of the alarms in the respective set. However, the performance of the AOI is highly dependent on the termination criteria and the abstraction hierarchies that need to be provided beforehand. Another use case may be a signature generation for traffic. Incoming traffic can be clustered by identifying dominant characteristics and a related pattern for each cluster is derived. Then, the derived patterns can be used for filtering traffic.

b) Attack mitigation: Mitigation mechanisms for DoS attacks can be distinguished into three groups: The first group consists of traditional approaches that use appliances like firewalls, intrusion prevention, and detection systems. They are deployed at fixed network positions with a fixed capacity. In other words, they rely on predefined policies and not able to make run-time decisions on their own. Thus, they will fail to handle unknown attacks, as well as the attacks, exceed their capacity [14]. The second group is cloud-oriented approaches that come with privacy drawbacks and increase the network latency due to a re-routing via the cloud [15]. By considering those issues, network operators have two options, namely to deploy appliances in a distributed or centralized manner. In the distributed scenario, high volume defense appliances like firewalls that can handle each type of known attacks can be deployed at each aggregation points. The centralized scenario provides deploying fewer appliances at central locations by rerouting traffic towards the center [16]. Although the second scenario seems more cost-effective, there are some critical issues to be addressed. Such a centralized approach requires rerouting the traffic explicitly, which causes latency and additional communication overhead. Also, the traffic can be monitored by third parties, which violates the privacy of the users [17].

The advent of SDN and NFV has brought a new perspective for network defense [18]. Key features of such technologies like global network view and flexibly positionable security functionality can be used for mitigating DDoS attacks. SDN can be used for enforcing a security policy by instructing forwarding entities to behave accordingly and NFV can be used for deploying Virtual Network Functions (VNFs) to propagate DDoS traffic from the victim network towards the source. The authors of [19]propose a DDoS attack mitigation framework including a pushback mechanism using SDN. The paper focuses on how to share attack information with the corresponding SDN controller, assuming the attack detected and identified. However, the proposed framework does not include a method to extract attack patterns.

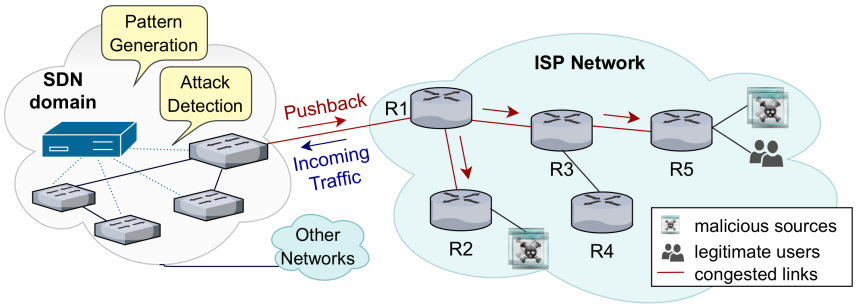


Fig. 1: Attack mitigation with pushback mechanism

III. SDN/NFV Based Collaborative Attack Mitigation

In this section, we introduce a collaborative SDN/NFV-based DDoS mitigation framework. For that, we first describe the overall framework and then we explain how we generate the attack signature and how we pushback the attack in detail.

A. *Overall architecture of SDN/NFV based Pushback*

The proposed framework which is shown in Figure 1 includes different components. SDN specific features facilitate the detection of attacks and take related mitigation actions on time. SDN controller can detect an attack at the ingress switch, which is S3, by analyzing the incoming traffic. Our intention in this paper is not to propose a new DDoS detection algorithm. Therefore, we select an entropy-based attack detection that is widely used in literature [20]. Entropy measures the randomness of flows during the given time window. If the randomness increases then the entropy also increases. In our case, if the number of the packets that have the same destination IP and destination port will increase quickly, entropy decreases sharply. Once the attack has been detected, traffic will be analyzed further for generating attack patterns which are significantly important as not to drop legitimate traffic (e.g., coming along R5). The pattern generation module derives an OF rule that describes the attack traffic. Then, the controller pushes the derived rule in S3 to filter unwanted traffic in the first phase of the attack mitigation. In case that the attack exceeds the capacity of the S3, the pushback mechanism will be triggered and collaborative attack mitigation will be initialized. The derived rule is propagated from R1 to R2 and R3, and subsequently to R5. Informed routers start to filter traffic so that, the attack traffic will be filtered before it arrives to the victim. In some cases, like collaborating with a non-SDN network, we might bring this filtering functionality by initializing the VNFs. Such a collaborative mitigation approach aims to make network resources available for the use of legitimate users by limiting the propagation of the attack traffic.

B. *Pattern Generation for Deriving Filtering Rules*

Attackers can easily spoof the IP addresses thus; we use Autonomous System Numbers (ASNs) instead IPs. We define our patterns using five attributes as source ASN (srcASN), source port (srcPort), destination ASN (dstASN), destination port (dstPort), and protocol (prot). Then, we apply the AOI algorithm on incoming network traffic to generalize it and create clusters of similar traffic flows. We use generalization hierarchies as shown in Figure 2. Port numbers are generalized three categories according to the range. We categorize IP addresses by AS membership. Then, protocols are categorized as TCP or UDP.

After the generalization, we select the pattern of the biggest cluster as an attack signature and transform the derived signature to IPtables rules that will be used for filtering traffic in the next step. For instance, assume that the pattern generated by AOI is shown by (a) in Figure 3. This can easily transform into a filtering rule which is shown by (b). Basically, it drops the packets coming from https port, which is 443, and designated to a registered port, which is defined in the range of 1024 to 49151. The source ASN is not assigned, therefore, we do not take it into account. There may exist more than one IP address located in the given destination AS. This mapping could be stored in the SDN controller and related IP addresses can be extracted. This translation can be done fast. Now, we have a filtering rule and we describe how we can use this rule for mitigating DoS attacks in the next paragraphs.

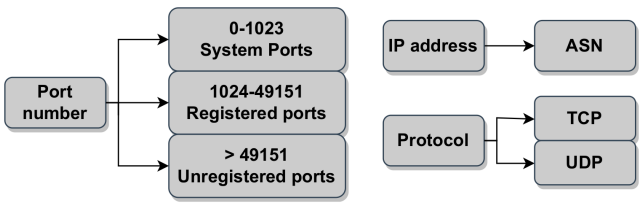


Fig. 2: Generalization Hierarchies

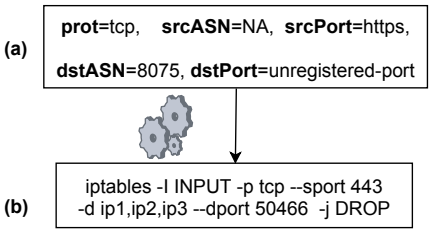


Fig. 3: Transforming pattern to filtering rule example

C. *Mitigating Attack by SDN/NFV-based Pushback*

To efficiently mitigate identified DDoS attacks, we use the Pushback mechanism. It protects networks from severe congestions due to a rapid increase in traffic as a result of a DoS attack. Although the Pushback is a mechanism known from the state of the art, we empower it with the use of SDN and NFV and with a novel scheme to distinguish DDoS and legitimate traffic.

Large networks may contain multiple interconnected local networks, controlled by a single SDN controller. The attack may aim to block the communication between local networks as well as the communication between controller and entities within the network. In such a scenario, the traffic increase at the switch can be detected via the responsible SDN controller. This can be done by the *Monitoring* module as a result of the periodic flow-stats request/reply mechanism as shown in Figure 4. The *Attack Detection* submodule analyses the collected traffic. In case of attack has been detected, an attack signature is generated and translated to a filtering rule. The derived rule is pushed to the respective switch via the OF. If the traffic exceeds the capacity of the switch, the *Attack Mitigation* submodule has been triggered to initialize the Pushback mechanism. For the Pushback, it might be necessary to use NFV to deploy firewall VNFs to filter attack traffic outside of the SDN domain. For that, the NFV module initializes the VNF instances to deploy the filtering rule in upstream routers. The controller pushes the related forwarding rules to re-route the attack traffic between the VNFs. Thus, the attack traffic is dropped iteratively in each router close to the attacker.

As long as we are in the SDN domain, NFV is not necessary for the Pushback. Assuming a scenario in which multiple SDN domains are connected, the extracted attack information is shared with the neighboring network's controller. The receiving controller validates the source of information by checking an additional certificate to authenticate the legitimacy of the transmitting controller. Then, the related rules will be installed at the connected nodes depending on the attack information. As a result, malicious flows are dropped. All neighboring SDN controllers perform the same procedure. In this way, DDoS attacks can be filtered collaboratively.

IV. Evaluation

In this section, we briefly outline the evaluation setup and metrics to evaluate the proposed pattern generation method, AOI, and LMP. We measure the quality in terms of attack and legitimate drop rates, and the performance in terms of CPU time consumption. Then, we briefly introduce the datasets that we used in our evaluation. Finally, we give our evaluation results.

A. *Evaluation setup and metrics*

We implemented the AOI and LMP algorithms in Python. We use the Pyasn module that enables offline and historical ASN lookups using BGP archives. Attack detection in our implementation based on entropy as in [25]. We use the pattern generation approach as introduced by us in Section III-B.

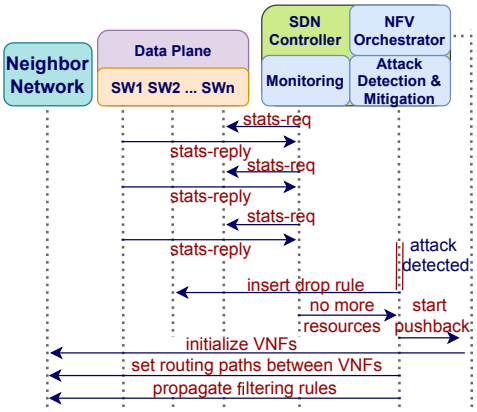


Fig. 4: Timing sequence diagram for our framework

We measure the strength of a defense mechanism with following metrics depending on how well it can detect and how fast does it react against attacks [26]:

- Classification Rate (CR): Ratio of the correct outcomes of the detection mechanism (true positives and true negatives) over the total outcomes.
- Recall: The ratio of true positives over the total desired positive outcomes.
- False Positive Rate (FPR): It is the ratio of the number of negatives wrongly categorized as positive over the total number of actual negatives.

B. *Datasets*

We evaluate our approach on four datasets that we briefly summarize in Table I. All datasets have attack and legitimate traffic data while background traffic is also included in CTU-13 dataset. Although there are several attacks in these datasets, we are only interested in the DoS attacks. Our aim is to show AOI can differentiate attacks independent from the attack type or dataset, while LMP can work well only for multi designated attacks. We run our experiments for NTP amplification DoS, ICMP-UDP-HTTP DDoS, DoS slowloris, DoS Slowhttptest, DoS Hulk, DoS GoldenEye, and IRC Botnets.

C. *Results*

The focus of this paper is to derive filtering rules efficiently and to use these rules to mitigate the attacks via SDN/NFV. For that, we conduct a set of experiments to investigate the performance of our framework. In our experiments, we extract patterns for different DDoS attack scenarios and investigate to which extent we can differentiate attacks from legitimate traffic. Our evaluation results for the different DDoS datasets (cf. Section IV-B) are shown in Table II. The CR shows the rate of correctly identified attack and legitimate traffic. Looking only at CR may be misleading, because it does not say anything about the individual classes. One class could be perfectly classified, while the other class is completely misclassified. Thus, we computed also Recall and FPR. The Recall metric shows how much of the DoS traffic has been detected. The FPR summarizes how much of the legitimate traffic has been classified as DoS.

Dataset	Time	Description	Attacks	DoS type
CICDDoS [21]	01/2019	To resemble real-world data, some human behavior models have been used for generating naturalistic legitimate background traffic	Reflection-based DDoS attacks: PortMap, NetBIOS, LDAP, MSSQL, UDP, UDP-Lag, SYN, NTP, DNS, SNM	one-to-one
CICIDS [22]	07/2017	Contains most up-to-date common attacks by resembling true world real-world data	Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS	mixed
ISCXIDS [23]	06/2010	Generated by analyzing real traces to create agent profiles. Created profiles are used in a testbed environment to generate the resulting dataset.	Brute Force SSH, HTTP DoS, DDoS by IRC Botnet	many-to-one
CTU-13 [24]	08/2011	Captured on the main router of the CTU University and contains a capture of several botnet scenarios	Different botnet scenarios including UDP-ICMP DDoS	mixed

TABLE I: Summary of used datasets

In the CICIDS dataset, most attacks have been carried out by multiple sources attacking a single victim. Since there is only one victim, it is easy to drop all packets targeting that destination. As can be seen from the results, both algorithms have 1.0 Recall value, which means that they drop all DoS packets. However, this may cause dropping legitimate traffic to the same IP as well as it happens with LMP. This can be seen from the CR; it differs significantly due to the miss-classified legitimate traffic. Also from the FPR, AOI only drops 1.20% of the legitimate traffic while LMP drops 84.56%. This shows that even though both algorithms can efficiently classify DoS, LMP is not able to differentiate legitimate and attack traffic.

The CTU dataset contains background traffic in addition to legitimate and attack traffic. Thus, the Recall value for LMP is lower than for the other datasets, because the extracted attack signatures are highly affected by background and legitimate traffic. Thus, DoS packets may not get dropped. Also, when dropping traffic that shares the same destination IP, a high portion (74.67%) of the legitimate traffic is lost as well.

For the CICDDoS dataset, LMP seems to perform slightly better than AOI in terms of the CR. However, LMP fails to differentiate legitimate traffic from attack traffic. While it drops 2% more attack traffic than AOI, it drops 39.41% of the legitimate traffic and AOI only 4.73%. Such a high legitimate drop rate derived from the distribution of the source and destination IPs. In the dataset, only one IP is under attack, which means that LMP decides to drop all the traffic that goes to this IP. In this case, legitimate traffic to this IP will be also dropped. However, AOI uses five attributes for filtering, so that it saves 95.27% of the legitimate traffic from being dropped.

In the ISCXIDS dataset, there is a big difference in CR between AOI and LMP, which is caused by miss-classified legitimate traffic. The Recall values indicate that both algorithms are able to detect nearly all attack traffic. However, LMP drops 60.56% of legitimate traffic while AOI only drops 22.70%, which is still much better than LMP.

Our experimental results reveal that AOI and LMP efficiently detect DoS traffic, but LMP falls behind in differentiating legitimate from DoS traffic. In all datasets, LMP efficiently detects attack traffic, because DDoS attacks generally target one or a set of IPs. Since LMP only looks at destination IPs, it is able to drop most attack traffic, but also a large part of legitimate traffic as well. AOI performs better here, as it uses multiple attributes instead of only one attribute as in LMP. Thus, AOI allows for more fine-grained filtering and differentiation of attack and legitimate traffic.

Finally, we focus on improving the performance of AOI. For that, we implement two different packet sampling approaches namely Systematic sampling (S1), which takes each kth sample, and Random sampling (S2), which selects packets with 1/k probability as explained in [27]. Then, we run experiments to see it can we improve the runtime of the pattern generation without sacrificing accuracy. We measure FPR and Recall values in dependence on different values of k as shown in Figure 5. S1 achieves lower FPR than AOI up to a certain point. After that point, the FPR starts to increase while the Recall values decrease for S1. In other words, S1 achieves better performance for low values of k. However, it cannot differentiate legitimate traffic as accurately as previous for the higher values of k. Unlike S1, the results of S2 are slightly inferior from AOI even for smaller k values. Since it collects random samples, we repeated the experiment 20 times for avoiding outliers. Based on the results, S2 can be used to achieve nearly the same classification accuracy as pure AOI.

We showed the detection performances of the algorithms, but security does not come free. We also need to consider the performance in terms of CPU time consumption. For that, we plot the CPU time consumption of both algorithms in dependence on an increasing number of packets in Figure 6. We investigate how the response time of the algorithm changes with an increasing number of packets. LMP is faster than AOI as expected, because it only takes the destination IP into account for filtering. Even though AOI makes several ASN lookups and filters based on five attributes, it can extract

	Attribute Oriented Induction				Longest Matching Prefix		
Dataset	CR	Recall	FPR		CR	Recall	FPR
CICIDS	0.9955	1.0	0.012		0.6900	1.0	0.8456
CTU-13	0.9988	0.9995	0.018		0.7063	0.7265	0.7467
CICDDoS	0.9747	0.9748	0.0473		0.9979	0.9991	0.3941
ISCXIDS	0.9036	0.9958	0.2270		0.7471	0.9958	0.6056

TABLE II: Evaluation of AOI and LMP algorithms with varying attack types

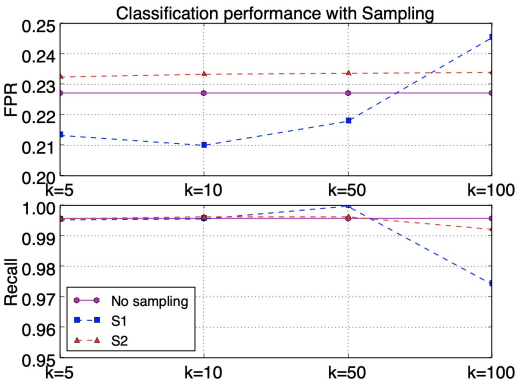


Fig. 5: Classification performance after sampling

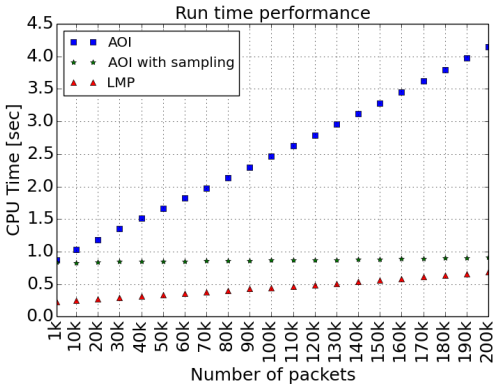


Fig. 6: CPU time consumption of pattern generation

attack signatures in a few seconds which is still reasonable. Our results indicate that sampling-based AOI, achieves better performance and stays close to LMP. Therefore, AOI with sampling achieves a high classification accuracy and runs close to real-time as LMP.

V. Conclusions

In this paper, we propose a collaborative DDoS mitigation framework by leveraging SDN/NFV technologies. To derive a pattern for filtering attack traffic, we adapt Attribute-Oriented Induction (AOI) in this domain. Our two-phased mitigation framework first tries to handle attacks locally. In case that huge volume attack that exceeds the capacity, pushback is initiated. We also describe how the attack is pushed back by NFV in non-SDN domains. An attack can be filtered close to the source of the attack and network resources are saved from being wasted. The evaluation on different datasets indicates that attack traffic can be differentiated from legitimate traffic by using a simple AOI generalization scheme. Furthermore, the results show that the most popular method from the state of the art, LMP, performs poor and drops high portions of the legitimate traffic. At the same time, our AOI scheme on top of sampled packets allows for a more accurate distinction of attack and legitimate traffic at low additional performance overhead.

References

[1] A. Shameli-Sendi, M. Pourzandi, M. Fekih-Ahmed, and M. Cheriet, "Taxonomy of ddos mitigation approaches for cloud computing," *Journal of Network and Computer Applications*, vol. 58, pp. 165–179, 2015.

[2] Y. Chen and K. Hwang, "Collaborative detection and filtering of shrew DDoS attacks using spectral analysis," *Journal of Parallel and Distributed Computing*, vol. 66, no. 9, pp. 1137–1151, 2006.

[3] G. Carl, G. Kesidis, R. R. Brooks, and S. Rai, "DoS attack-detection techniques," *IEEE Internet computing*, vol. 10, no. 1, pp. 82–89, 2006.

[4] B. Rashidi and C. Fung, "Cofence: A collaborative DDoS defence using NFV," in *CNSM*. IEEE, pp. 160–166.

[5] A. Jakaria, W. Yang, B. Rashidi, C. Fung, and M. A. Rahman, "Vfence: A defense against DDoS attacks using NFV," in *COMPSAC*. IEEE, 2016, pp. 431–436.

[6] C. J. Fung and B. McCormick, "Vguard: A DDoS attack mitigation method using NFV," in *CNSM*. IEEE, 2015, pp. 64–70.

[7] L. Zhou and H. Guo, "Applying NFV/SDN in mitigating ddos attacks," in *TENCON*. IEEE, 2017, pp. 2061–2066.

[8] M. Stein, M. Fischer, I. Schweizer, and M. Mühlhäuser, "A classification of locality in network research," *ACM Computing Surveys (CSUR)*, vol. 50, no. 4, pp. 1–37, 2017.

[9] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Aggregate-based congestion control," *Computer Communication Review*, vol. 32, no. 3, 2002.

[10] J. Ioannidis and S. M. Bellovin, "Implementing pushback: Router-based defense against DDoS attacks," 2002.

[11] Y. Cai, N. Cercone, and J. Han, "Learning in relational databases: an attribute-oriented approach," *Computational Intelligence*, vol. 7, no. 3, pp. 119–132, 1991.

[12] J. Han, Y. Cai, and N. Cercone, "Knowledge discovery in databases: An attribute-oriented approach," in *VLDB*, vol. 18, 1992, pp. 574–559.

[13] K. Julisch, "Clustering intrusion detection alarms to support root cause analysis," *TISSEC*, vol. 6, no. 4, pp. 443–471, 2003.

[14] C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks*, vol. 44, no. 5, pp. 643–666, 2004.

[15] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and R. Buyya, "DDoS attacks in cloud computing: Issues, taxonomy, and future directions," *Computer Communications*, vol. 107, pp. 30–48, 2017.

[16] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic DDoS defense," in *USENIX*, 2015, pp. 817–832.

[17] T. Alharbi, A. Aljuhani, and H. Liu, "Holistic DDoS mitigation using NFV," in *CCWC*. IEEE, 2017, pp. 1–4.

[18] C. C. Machado, L. Z. Granville, and A. Schaeffer-Filho, "Answer: Combining NFV and SDN features for network resilience strategies," in *ISCC*. IEEE, 2016, pp. 391–396.

[19] S. Hameed and H. Ahmed Khan, "Sdn based collaborative scheme for mitigation of DDoS attacks," *Future Internet*, vol. 10, no. 3, p. 23, 2018.

[20] S. Behal and K. Kumar, "Detection of DDoS attacks and flash events using information theory metrics–an empirical investigation," *Computer Communications*, vol. 103, pp. 18–28, 2017.

[21] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic ddos attack dataset and taxonomy." IEEE, 2019.

[22] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." 2018.

[23] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.

[24] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, 2014.

[25] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed DDoS detection mechanism in SDN," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 310–317.

[26] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against DDoS flooding attacks," *IEEE communications surveys & tutorials*, pp. 2046–2069, 2013.

[27] D. Tammaro, S. Valenti, D. Rossi, and A. Pescapé, "Exploiting packet-sampling measurements for traffic characterization and classification," *International Journal of Network Management*, vol. 22, no. 6, pp. 451–476, 2012.

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