NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENT OF ELECTRICAL AND COMPUTER ENGINEERING

RODRIGO GONÇALVES BORGES SIMÕES

Licenciate in Electrical and Computer Engineering

INTELLIGENT FLOW SCHEDULLING IN DETERMINISTIC NETWORKS

MASTER'S IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon

September 2023

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENT OF
ELECTRICAL AND COMPUTER ENGINEERING

INTELLIGENT FLOW SCHEDULLING IN DETERMINISTIC NETWORKS

RODRIGO GONÇALVES BORGES SIMÕES

Licenciate in Electrical and Computer Engineering

Adviser: Pedro Miguel Figueiredo Amaral
Associate Professor, NOVA School of Science and Technology

Examination Committee:

Chair: Arnaldo Manuel Guimarães Batista
Associate Professor, *NOVA School of Science and Technology*

Rapporteurs: Luís Filipe Lourenço Bernardo
Associate Professor, *NOVA School of Science and Technology*

Adviser: Pedro Miguel Figueiredo Amaral
Associate Professor, *NOVA School of Science and Technology*

MASTER'S IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2023

Intelligent Flow Scheduling in Deterministic Networks

Acknowledgments

I wish to express my gratitude to my mentor, Pedro Amaral, for both proposing and guiding this dissertation. I would also like to extend my appreciation to all the professors and teaching assistants who have been an integral part of my academic journey. Their accessibility and unwavering support proved essential in navigating the challenges of this Master's degree program as smoothly as possible.

I am deeply thankful to the NOVA School of Science and Technology as an institution, which has consistently prioritized the development of its students, encompassing not only academics but also various aspects of life.

This academic journey would not have been as enriching without the friends and colleagues who shared it with me. From collaborative group projects to friendships that will last a lifetime, I want to extend my gratitude to each and every one of them. A special mention goes to the self-proclaimed "Andiamo Fratello" group, which provided me with memories and moments of fun through even the most stressful times. I am very thankful to Patricia Neves, a friend and partner who, in both the highs and lows, always motivated me to strive for excellence.

A heartfelt thank you goes out to my family, an unwavering source of support and guidance that never allowed me to falter. This work represents the culmination of five years of collective and individual effort. I hope it reflects the degree of excellence that this institution, my friends, and my family have consistently instilled in me.

"No research without action,
no action without research"
(Kurt Lewin).

ABSTRACT

Intelligent flow scheduling in deterministic networks refers to the process of optimizing the allocation of network resources to data flows in a deterministic manner based on various parameters and criteria. In a deterministic network, each data flow is guaranteed a specific maximum delay budget, which ensures that the flow's performance is predictable and consistent.

To achieve this, intelligent flow scheduling uses algorithms and protocols that dynamically allocate resources based on real-time network conditions. These algorithms may use techniques such as packet scheduling, traffic shaping, and congestion control to ensure that network resources are allocated fairly and efficiently.

This is particularly important in applications such as real-time streaming, video conferencing, and online gaming, where delays and interruptions can be extremely disruptive. Additionally, it can help to reduce overall network traffic and improve security by controlling the flow of data through the network.

For this dissertation a Deep Reinforcement Learning approach was taken to Implement intelligent flow scheduling which involved training an agent to make decisions on how to allocate priorities for flow transmission based on various parameters such as flow packet size, and latency requirements. The agent interacts with the network by observing the current network state and selects actions to allocate resources to data flows. The agent then receives feedback on the performance of the network, which is used to adjust its decision-making process.

A network simulation was conducted using OMNET++ and INET to model a network hosting six applications: four operating with deterministic requirements and two utilizing best-effort traffic. The network switches were equipped with an Asynchronous Traffic Scheduler known as UBS, which is currently employed by the Time Sensitive Networking Work Group for asynchronous networks.

In this setup, Deep Reinforcement Learning was employed to determine the optimal prioritization for each incoming deterministic flow. The decisions made by the DRL Agent influenced how the Asynchronous Traffic Scheduler treated the flow and, consequently, its performance within the network. Each deterministic flow had specific delay requirements, and the DRL Agent's success metric was the number of packets in each flow that achieved an end-to-end delay lower than their specified requirements.

The DRL Agent effectively optimized resource allocation, resulting in a 42% reduction in the number of packets exceeding their deterministic delay requirements upon arrival, when compared to a standard fixed prioritization solution.

Keywords: DRL, Intelligent flow scheduling, Deterministic Networks, Quality of Service

Resumo

O agendamento inteligente de fluxos em redes determinísticas refere-se ao processo de otimização da atribuição de recursos a fluxos numa rede de forma determinística com base em vários parâmetros e critérios. Numa rede determinística, a cada fluxo de dados é garantido um *budget* específico de *delay* máximo, que assegura que o desempenho do fluxo é previsível e consistente.

Para o conseguir, o agendamento inteligente de fluxos utiliza algoritmos e protocolos que alocam dinamicamente recursos com base em condições da rede em tempo real. Estes algoritmos podem utilizar técnicas tais como agendamento de transmissão, modelação de tráfego, e controlo de congestionamento para assegurar que os recursos de trabalho em rede sejam atribuídos de forma justa e eficiente.

O agendamento inteligente de fluxos é uma abordagem interessante para ajudar a melhorar o desempenho, a fiabilidade e a eficiência das redes, otimizando a atribuição de recursos aos fluxos de dados. Ao assegurar que cada fluxo recebe os recursos de que necessita, é possível evitar congestionamentos e atrasos, e ajudar a manter uma experiência previsível e consistente dos utilizadores. Isto é particularmente importante em aplicações como o *streaming* em tempo real, onde os atrasos e interrupções podem ser extremamente prejudiciais. Adicionalmente, pode ajudar a reduzir o tráfego global da rede e melhorar a segurança através do controlo do fluxo de dados através da rede.

Para esta dissertação foi utilizada uma abordagem que utiliza DRL para realizar o agendamento inteligente de fluxos e implica o treino de um agente que tome decisões sobre como atribuir uma prioridade de transmissão ao fluxo com base em vários parâmetros tais como tamanho de pacotes, e requisitos de *delay*. O agente interage com o ambiente da rede, observa o estado atual da rede, e seleciona ações para atribuir recursos aos fluxos de dados. O agente

recebe feedback sobre o desempenho da rede, que é utilizado para ajustar o seu processo de tomada de decisão.

Foi realizada uma simulação de rede utilizando o OMNET++ e o INET para modelar uma rede com seis aplicações: quatro que funcionam com requisitos determinísticos e duas que utilizam tráfego *best effort*. Os *switches* de rede estavam equipados com um *shaper* de tráfego assíncrono conhecido como UBS, que é atualmente utilizado pelo Time Sensitive Networking Work Group para redes assíncronas.

Nesta configuração, Deep Reinforcement Learning foi utilizado para determinar a priorização ideal para cada fluxo determinístico. As decisões tomadas pelo agente DRL influenciam a forma como o *shaper* de tráfego assíncrono trata o fluxo e, consequentemente, o seu desempenho na rede. Cada fluxo determinístico tinha requisitos específicos de *delay*, e a métrica de sucesso do agente DRL é o número de pacotes em cada fluxo que atinge um valor de *delay* inferior aos requisitos especificados.

O agente DRL otimizou a alocação dos recursos, resultando numa redução de 42% no número de pacotes que excederam os requisitos de *delay* determinístico, quando comparado com uma solução de prioridades fixas.

Palavras-chave: DRL, Agendamento Inteligente de Fluxos, Redes Deterministicas, Qualidade de serviço

CONTENTS

List of Figures

List of Tables

ACRONYMS

TSN	Time Sensitive Network
DRL	Deep Reinforcement Learning
AI	Artificial Intelligence
ML	Machine Learning
QoS	Quality of Service
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
SDN	Software Defined Network
RL	Reinforcement Learning
PHY	Physical Layer
ANN	Artificial Neural Network
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
MAC	Medium Access Control
MDP	Markov Decision Process
DQN	Deep Q Network
5G	5th Generation of Wireless Technology
JSON	JavaScript Object Notation

FIFO	First-in, First-Out
GUI	Graphical User Interface
DetNet	Deterministic Network

1

INTRODUCTION

1.1 Motivation

Conventional networks characteristically tend to have difficulty in reducing end-to-end latencies to only tens of milliseconds [1] which is becoming a requirement for modern applications that require ultra-low latency for various industrial, healthcare, transportation, and communication scenarios. ULL demands vary from a few microseconds to milliseconds for industrial applications, 1 millisecond for the tactile internet, and around 100 microseconds for the one-way fronthaul in wireless cellular networks [2] [3]. These requirements are essential for real-time connectivity for tele-surgery, transportation, autonomous automotive vehicles [4], augmented and virtual reality, and robotic applications. Therefore, a dedicated mechanism that can cater to diverse ULL requirements in heterogeneous environments and applications would be highly beneficial [5].

ULL applications require low latency, high bandwidth, and consistent network performance to deliver a seamless user experience. As a result, there is a growing demand for networking technologies that can guarantee the required QoS metrics and meet the stringent performance requirements of these applications. The proliferation of emerging technologies such as the Internet of Things and autonomous vehicles has also intensified the need for these types of networking technologies.

Deterministic Networks have emerged as a promising approach to meet these demands by offering predictable performance and enhanced Quality of Service guarantees. However, optimizing such networks for providing deterministic guarantees to applications that require it without harming other applications with prioritization and scheduling mechanisms poses significant challenges that require innovative solutions. Deterministic networks aim to provide

predictable and bounded performance by eliminating uncertainties in the transmission of data packets. These networks leverage time sensitive networking protocols, scheduling algorithms, and deterministic communication techniques to achieve low latency, high reliability, and efficient resource utilization.

Advancements in hardware technologies, such as faster processors, high-capacity memory, and high-speed communication interfaces, have also facilitated the development of deterministic networking.

The motivation for conducting this dissertation lies then in the need for efficient networking solutions and the technological advancements in deterministic networks and DRL. By addressing the challenges of flow scheduling and prioritization in deterministic networks, it will be possible to contribute to the development of networking technologies that can support the ever-increasing demands of data-intensive applications and ensure seamless connectivity in various domains.

Also, the potential societal impact of optimizing flow allocation in deterministic networks is significant. Industries such as healthcare, finance, and transportation heavily rely on real-time data transfer for critical operations. By optimizing network performance, we can enhance the delivery of healthcare services, enable faster financial transactions, and support efficient transportation systems. The outcomes of this research have the potential to positively impact various sectors, improving productivity, reliability, and user experiences.

This research also aligns with the broader trend of leveraging DRL and machine learning techniques in networking. As AI continues to revolutionize various industries, its application in networking is crucial for keeping pace with the demands of the digital era. DRL is relevant for trying to optimize control problems where it is not possible to create a reliable model of a system that works with all possible situations. By exploring the potential of DRL for flow scheduling and prioritization optimization, we contribute to the ongoing research efforts in AI-driven networking and pave the way for future innovations in this domain.

1.2 Problem Statement

Optimizing flow allocation in deterministic networks presents unique challenges. The complex nature of network traffic patterns, varying flow demands, and the need for efficient resource allocation require intelligent decision-making mechanisms. AI approaches to flow scheduling have proven to have the capability of being superior to traditional approaches

based on static configurations or manual optimization techniques. These will be addressed in chapter 2.

In this dissertation the focus will be on optimizing Intelligent flow scheduling for Asynchronous deterministic networks, since It's an area of research that is emerging and in need of Innovative solutions to be applied to real world problems. To achieve this the problem to address Is the allocation of priorities to the flows in a manner that allows for the best possible use of resources in order to guarantee that the delay requirements of each deterministic flow are met.

The metric of success when attempting to solve this issue is the number of packets that achieve delay requirements in the deterministic flows that operate in a given network, therefore it is necessary to build a solution that not only focuses on minimizing delay but on maximizing the utilization of network resource in order to avoid that any packet of a deterministic flow arrives over the specified end-to-end delay maximum.

1.3 Proposed solution

This dissertation aims to tackle the problem described, which involves determining the best configuration for assigning priorities to the incoming flows of a network. The focus is on Asynchronous IEEE 802.1 TSN networks, which currently uses IEEE 802.1Qcr Asynchronous Traffic Shaper as the main solution developed by the TSN Working Group for asynchronous networks. The scheduler used to optimize the Asynchronous scheduling is called Urgency-Based Scheduler and works to ensure per-flow deterministic QoS guarantees, such as zero packet loss and bounded latency, in a practical way. An In-depth analysis of these scheduler and its performance will be done in chapter 2.

A DRL-based solution is proposed. This solution utilizes DRL and ATS performance models to ensure the predictability of flows delay. DRL is up and coming approach to different types of networking problems and although its use for scheduling is recent there have been some promising results.

DRL combines deep learning and reinforcement learning to train agents that can make intelligent decisions based on environmental feedback. By training a DRL Agent to optimize flow allocation in deterministic networks, we hope to leverage its ability to learn from experience, adapt to network dynamics, and optimize performance metrics. The application of DRL in networking opens up new possibilities for enhancing the efficiency, reliability, and scalability of flow allocation algorithms.

The use of DRL with the Asynchronous Traffic Shaper will be based on an action of the DRL Agent proposed that assigns the best possible priority for any given deterministic flow that arrives in a network in order to minimize the number of packets that fail to fulfill the delay requirements of all deterministic flows in the network. This priority then determines how the flow is handled by the Asynchronous Traffic Shaper.

1.4 Document structure

The state-of-the-art chapter provides an overview of the latest advancements in time-sensitive networking technology as well as a discussion of the most relevant research to date concerning the use of AI and DRL to tackle networking challenges.

The document then includes a chapter on the network simulation, which provides a detailed description of the simulation environment utilized for testing and evaluating network performance, as well as the implementation of an asynchronous scheduler called UBS and the configuration of the network's behavior and visibility. The simulation environment serves as a crucial platform for conducting experiments and assessing the performance of the DRL Agent.

Another chapter tackles the behavior, implementation, and configuration of different aspects of the DRL Agent providing a detailed description of parameterization and algorithms used.

A chapter on simulation and results follows, presenting the results of the simulation experiments, including a comparison of optimal baseline network performance, and network performance with the DRL Agent.

Finally, the document concludes with the chapter on conclusions and future work, which summarizes the findings of the research and provides recommendations for future research directions.

2

STATE OF THE ART

This chapter provides an overview of the current status of research and development in the field of deterministic networking, flow scheduling and Deep Reinforcement Learning.

2.1 Deterministic Networking

The domain of deterministic networking is dedicated to addressing the demands of real-time applications by ensuring minimal data loss rates, stable packet delay fluctuations, and predictable latency limits. This technology is being developed by different standard organizations to meet the requirements of deterministic applications and recently it is possible to see a significant increase in the number of research papers attempting to optimize and provide innovative solutions to different deterministic networking problems.

The main areas of work for deterministic working [6] can be divided into the following categories [5]:

- **Flow Synchronization:** In deterministic networking standards, a critical aspect is network-wide precise time synchronization. This entails establishing a shared time reference across all network entities. By achieving this common time base, data and control signaling can be scheduled optimally, ensuring synchronized and deterministic communication.
- **Flow Management:** Flow management aims to facilitate the exploration, configuration, monitoring, and reporting of the capabilities of bridges and end stations. Through specialized protocols for stream reservation, this capability facilitates efficient allocation of network resources and supports real-time communication requirements.

- **Flow Control:** In deterministic-enabled bridges, flow control governs how frames belonging to specific traffic classes are handled. This plays a vital role in addressing the scheduling challenges faced within the environment, ensuring that traffic is efficiently managed and delivered according to prescribed priorities.
- **Flow Integrity:** Flow Integrity consists of standardized techniques, including frame replication and elimination reliability, are employed to ensure robust and resilient data delivery.

2.1.1 Working groups

Working groups are collaborative teams within professional organizations that focus on specific projects or areas of interest. In the context of networking, the Institute of Electrical and Electronics Engineers (IEEE) has a working group called IEEE 802.1, which is responsible for developing and maintaining standards for local and metropolitan area networks.

In the context of the IEEE 802.1 Working Group, several areas of investigation and development have been identified, including the architecture of LANs/MANs, internetworking between different types of networks, security, overall network administration, and protocols above the MAC and LLC layers. The primary standard that the group has developed is IEEE 802.1Q [7], which specifies the protocols and architecture for communication between linked bridges, layers, and sublayers adjacent to the main 802.1 layer. The IEEE 802.1Q standard also includes eight traffic classes, which are used to give different priorities to flows traversing the network.

Within the IEEE 802.1 working group, there is a subgroup known as the Time-Sensitive Networking (TSN) working group [8]. TSN extends Ethernet technology to enable deterministic real-time communication over standard Ethernet networks. The TSN working group's main objective is to develop standards and protocols that allow time-sensitive applications to coexist with traditional data traffic on Ethernet infrastructure.

The TSN working group collaborates with other IEEE 802.1 working groups and coordinates with relevant IEEE working groups, such as IEEE 802.3 for Ethernet physical layer specifications and IEEE 1588 for clock synchronization. This collaboration ensures seamless integration of TSN standards with existing LAN/MAN standards, providing a comprehensive solution for deterministic real-time communication in Ethernet networks.

The TSN Working Group addresses the four key areas of deterministic networking requirements: time synchronization, bounded low latency, ultra-reliability, and dedicated resources. While this dissertation primarily focuses on bounded low latency, specifically

asynchronous traffic shaping, it is essential to acknowledge the significance of other protocols in optimizing network performance. Real-life implementation of Time-Sensitive Networks relies on the collaborative efforts of all these protocol areas.

In a TSN network, there are two types of devices: bridges and end stations. The fundamental components of a TSN network include talkers, which are responsible for producing streams, listeners, which serve as the destination of a stream, streams, which represent the flow of data from a talker to one or more listeners, and bridges, which are responsible for forwarding streams. The control and data planes of the bridge are separate, allowing for the control to be done with distributed protocols or an external agent such as an SDN controller.

TSN was created to address the needs for more deterministic networks on a Layer 2 level, however, the networks that have these needs are growing larger and already require deterministic forwarding beyond the LAN boundaries. These networks require a new model which the IETF DetNet Working Group [6] addresses for the use of routed networks in deterministic applications, extending the TSN data and control plane into the Layer 3 domain. Some interesting use-cases are professional audio and video, wireless for industrial applications and Network Slicing.

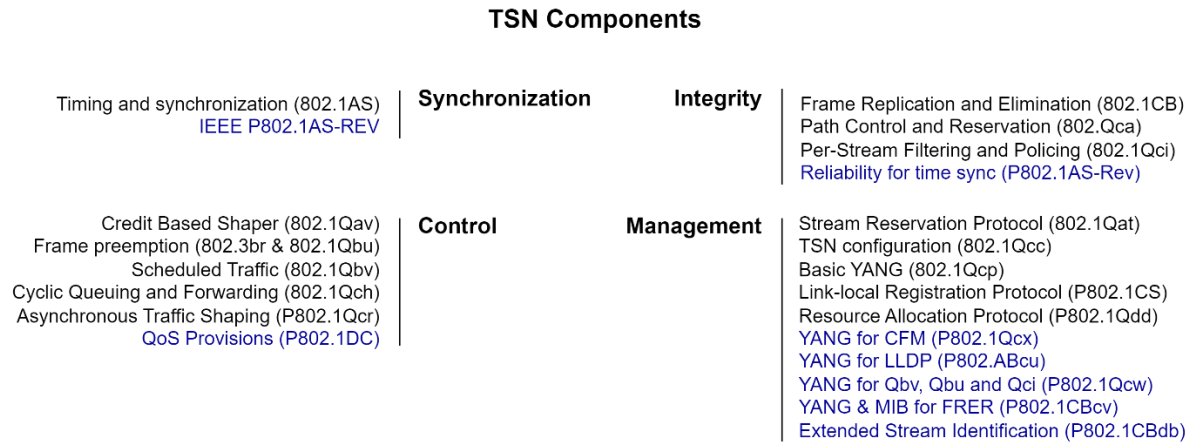

Figure 2-1 - Diagram of TSN Components

2.1.2 Deterministic networking synergies with 5G

5G promises to deliver ultra-fast speeds, low latency, high reliability, and massive connectivity.

To support ULL communications services in 5G wireless systems, the TSN and Deterministic Networking (DetNet) standards and research findings are likely to be heavily relied upon.

Deterministic networking's synergies with 5G can create new opportunities and benefits for both network operators and users. By combining deterministic networking and 5G, network operators can optimize their network resources and performance, reduce operational costs and complexity, and offer tailored services. End users can enjoy improved user experience, higher quality of service, and access to new applications and functionalities that require deterministic networking capabilities. Some examples of deterministic networking synergies with 5G are [9]:

- **Industrial automation:** Deterministic networking can enable real-time control and coordination of industrial processes and machines over 5G networks, ensuring high reliability, low latency, and high security.
- **Smart grid:** Deterministic networking can facilitate distributed generation, and demand response into the power grid over 5G networks, enabling efficient and resilient energy management and distribution.
- **Autonomous vehicles:** Deterministic networking can support the communication and cooperation of autonomous vehicles over 5G networks, enabling safe and efficient transportation and mobility.

2.2 Flow Control

In the realm of TSN, flow control standards and protocols specify a framework through which TSN enabled bridges decide how to treat each traffic class. The principle behind all flow control frameworks is one where TSN flows have privileges over non-TSN flows.

Flow scheduling refers to the process of organizing and managing the transmission of data packets within a network to ensure efficient and reliable data delivery. It involves determining the order and timing of data packet transmission, considering factors such as priority, bandwidth requirements, and network conditions. Flow scheduling is crucial in time-sensitive applications where meeting strict deadlines and minimizing latency are essential.

2.2.1 Time-Aware Shaper

IEEE 802.1Qbv Time-Aware Shaper (TAS) [10] was developed as a solution to the limitations of the IEEE 802.1Qav CBS mechanism. TAS is designed to cater to traffic with deterministic requirements and necessitates a network that is fully synchronized. TAS achieves this by scheduling priority traffic during specific time-triggered windows. To avoid any interference between best-effort and priority traffic, a guard band is defined before each window. However, this

approach introduces some delay because of the waiting time until the start of the next time-triggered window [11]. While TAS is more suitable for networks that require strict timing requirements, it doesn't offer much scalability due to the intricacy of synchronizing a network of moderate size.

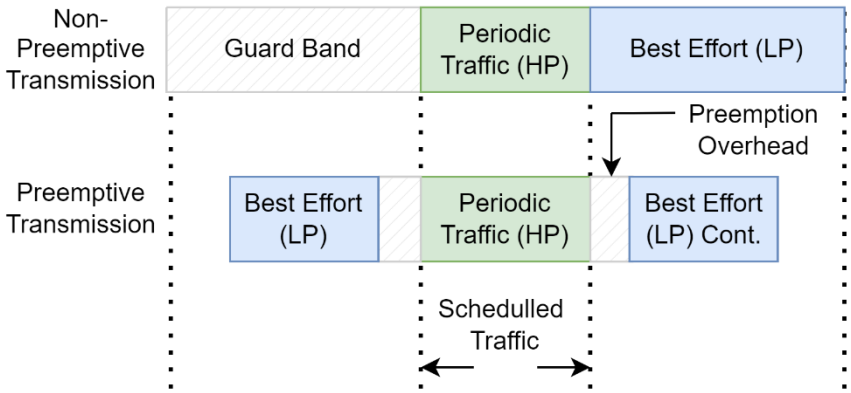

Figure 2-2 - Time-aware Shaper with frame preemption

To overcome the issue of low priority frames delaying the transmission of high priority frames, the 802.1 and 802.3 Task Groups introduced the concept of frame preemption [11]. The approach involves dividing the egress port of a bridge into two MAC interfaces, namely the express MAC and preemptable MAC. All frames are initially mapped to the eMAC, and frames that are identified as preemptable can be assigned to the pMAC and wait for the eMAC to finish transmitting high priority frames before transmitting. This ensures that high priority frames are not delayed by the transmission of lower priority frames, providing improved efficiency and reduced latency [5].

2.2.2 Cyclic Queuing and Forwarding

The TSN TG developed the Cyclic Queuing and Forwarding (CQF) standard to synchronize the queuing and forwarding operations across a TSN network with the aim of achieving zero congestion loss and bounded latency [12]. This standard requires that all bridges in the network are 802.1AS enabled and time synchronized. Time-sensitive streams are scheduled at each cycle, and the worst-case deterministic delay is two times the cycle time multiplied by the number of hops between the sender and receiver. This approach allows for achieving these objectives regardless of the network topology. To further reduce the cycle time, CQF can be used in combination with frame preemption [12].

2.2.3 Asynchronous Traffic Shaper

TAS relies on network-wide synchronization for its operation, which can limit its efficiency in utilizing available bandwidth. To address these limitations, asynchronous traffic shapers were developed.

As early as 1993 a discussion of asynchronous shaping benefits was occurring, in [13] the authors provide a definition of basic concepts for scheduling algorithms and provide experimentations that establish asynchronous schedules as something that can be obtained.

The IEEE 802.1Qcr Asynchronous Traffic Shaper was developed as a response to time-sensitive networking needs in asynchronous applications. In [14] the authors introduce the Urgency-Based Scheduler (UBS) and make an analysis of the worst-case latency achieved with this mechanism using different interleaved shapers. The same authors of [14] expand their study in [15] and try to solve the UBS synthesis problem which consists of the assignment of real-time flows to shaped queues and priority levels to queues at every hop along the predefined path. ATS operates on a per-hop basis, utilizing per-flow queues and prioritizing urgent traffic over non-urgent traffic using an urgency-based scheduler.

UBS [14] includes a two-level queuing system consisting of shared queues and shaped queues. Incoming flows are allocated to the shaped queues in accordance with three rules:

1. **QAR1:** frames from different senders are not allowed in the same queue.
2. **QAR2:** frames from the same sender without the same priority levels can't be on the same queue.
3. **QAR3**: frames from the same sender with the same priority in the sender but different priorities in the receiver are not allowed on the same queue.

The shared queues then merge all the shaped queues that belong to the same priority level.

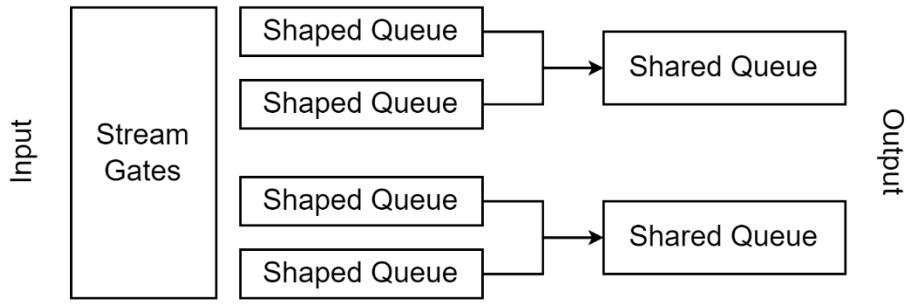

Figure 2-3 - UBS Queue model.

The Urgency-Based Scheduler differs from a strict priority approach having a separate queue per priority in two ways.

While UBS uses an urgency-based scheduling algorithm that considers both the priority level and the waiting time of packets in the queue and calculates an urgency value for each packet to select the next packet for transmission, the strict priority scheduling makes the decision by selecting packets from the highest priority queue for transmission.

While UBS provides flexibility in managing the transmission order based on both priority and waiting time and allows for dynamic adjustment of urgency levels, considering the current congestion and delay conditions in the network, the strict priority scheduling offers simplicity with each priority level having its own dedicated queue, and packets being transmitted in strict priority order.

The key difference between the two approaches lies then in the consideration of waiting time in the scheduling decision. UBS considers both priority and waiting time to determine the urgency of packets, which allows for more dynamic and adaptive scheduling. On the other hand, the separate queue per priority approach relies solely on priority-based scheduling without considering waiting time.

The UBS standard also proposes two interleaved shaping methods that won't be the focus of this dissertation, Length-Rate Quotient (LRQ) and Token Bucket Emulation (TBE). These approaches achieve rate limiting in the ATS mechanism, but they use different algorithms to do so. LRQ is based on a frame-by-frame leaky bucket algorithm, while TBE uses a token-based leaky bucket algorithm. The key difference between these two methods is that TBE allows for some burstiness in the output of packets, while LRQ provides a completely stable output [14].

a) Comparison with Paternoster

In [16] the authors compare the performance of UBS with a different ATS mechanism called Paternoster. The Paternoster algorithm is developed based on a cyclic scheduling approach, which offers deterministic and bounded delay while eliminating the need for synchronous timing. Simulation results were obtained to compare the performance of two different approaches and highlight the characteristics of the schedulers under varying input intensities. The UBS approach exhibits a higher average delay value and inefficient bandwidth utilization in lightly loaded situations. On the other hand, under situations with increased intensity the UBS approach performed much better than the Paternoster approach.

b) Comparison with TAS

In [17], a direct comparison between the Time-Triggered Traffic Shaper (TAS) and the Asynchronous Traffic Shaper (ATS) was conducted in the context of a typical industrial control ring network. The main focus of this comparison was to evaluate the mean and maximum packet delays, as well as packet losses, for both TAS and ATS, considering two types of traffic: random sporadic traffic and periodic traffic.

The ATS was evaluated using an approach slightly different from the standard, as depicted in Figure 2-4. The ATS shaper uses the eligibility time and the QoS currently experienced to direct traffic to the urgent queue. This approach is referred to as per-hop shaping. To ensure fairness, the regular Strictly Timed (ST) and Best Effort (BE) queues are multiplexed at the egress. It is important to note that all queues in the system operate under the First-In-First-Out principle.

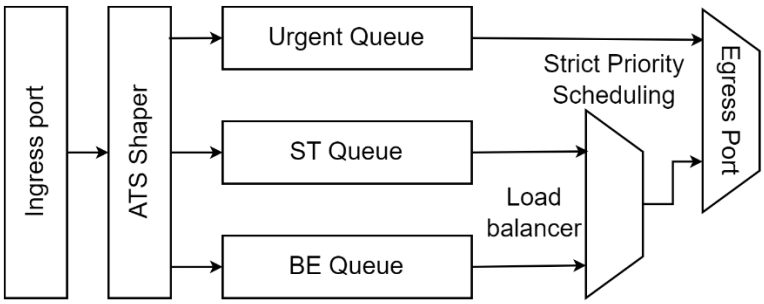

Figure 2-4 - Alternative ATS model

The evaluation results showed that, properly configured, TAS can achieve better performance for both sporadic and periodic traffic scenarios. On the other hand, ATS demonstrated

relatively good performance for sporadic traffic. However, it encountered challenges when dealing with moderate to high loads of periodic traffic, where its capabilities were not as effective.

2.2.3.1 Applications of deterministic ATS

In [18] the researchers conducted an evaluation of the maximum capacity of asynchronous Time-Sensitive Networking (TSN) networks to handle industrial traffic flows. Their objective was to assess the highest achievable utilization provided by asynchronous TSN. To conduct this assessment, the authors utilized UBS on three different types of industrial network topology: Star topology, Daisy chain topology, and Ring topology. The simulation results indicated that the Star topology exhibited the highest probability of flow rejection, meaning it struggled to accommodate industrial traffic flows efficiently. On the other hand, the Ring topology outperformed the Daisy chain topology, showing better performance in terms of accommodating traffic flows in the asynchronous TSN network.

In [9] the authors discussed the potential of Time-Sensitive Networking as a promising network technology to cater to the Ultra-Reliable Low-Latency Communication requirements in 5G. They emphasized the significance of TSN for enabling critical services to coexist with Mobile Broadband traffic in various scenarios. This article particularly explored the adoption of asynchronous TSN for 5G backhauling and its relevant aspects. The authors highlighted that per-hop shaping is a viable solution to tackle the issue of large worst-case delays associated with First-In-First-Out (FIFO) buffering. They also advocated for asynchronous networks over synchronous ones due to their lower complexity and better scalability. Additionally, synchronous networks were noted to have worse network resource utilization. The main contributions of this work were summarized as follows: firstly, providing a comprehensive overview of the key concepts related to asynchronous TSN networks. Secondly, discussing the flow allocation problem in networks based on Asynchronous Traffic Shaper (ATS) and presenting various approaches to address it. Thirdly, exploring the adoption of Machine Learning (ML) techniques for automating the management of TSN Backhaul Networks (BNs). The authors emphasized the importance of incorporating analytical performance models to ensure reliable decision-making by ML agents.

In [19] the authors proposed a stochastic solution for achieving deterministic networking in Industrial Wireless Networks (IWNs) using asynchronous scheduling. In industrial control and monitoring processes, it is crucial to have communications with deterministic guarantees of delay across the network. However, the challenge arises as a conflict exists between high reliability and a lower delay, necessitating a well-balanced scheduling algorithm in IWNs to strike the right compromise. The authors of this paper addressed this challenge by introducing a new scheduling model that supports asynchronous scheduling without time synchronization between the nodes between nodes. This model aims to meet the given delay requirement while ensuring probabilistic reliability. The objective was to design a scheduling algorithm that achieves a high ratio of packet delivery and Is able to satisfy deadline requirements with reliability. To achieve this, the authors devised a probabilistic algorithm based on the Monte Carlo tree search (MCTS) method. In terms of transmission reliability, the simulation results unequivocally demonstrate that the proposed algorithm, E-MCTS, significantly outperforms the classic Monte Carlo tree search algorithm.

The authors of [20] confront a challenge in the realm of 5G backhaul networks: the critical flow allocation problem. These networks are implemented as asynchronous TSN networks, utilizing the Asynchronous Traffic Shaper as a fundamental building block. To address this challenge, the authors introduce an offline solution named "NEPTUNO," short for "Next Generation Transport Network Optimizer." NEPTUNO takes a multifaceted approach, combining exact optimization techniques, heuristic strategies, and data analytics to comprehensively tackle the flow allocation problem. Its primary objective is to maximize the acceptance of data flows while ensuring that Quality-of-Service requirements for critical data streams are met. To assess the performance of NEPTUNO, the authors consider several crucial factors, including the level of optimality achieved, the time required for execution, and the rate of rejected data flows. The evaluation results are positive, revealing that NEPTUNO attains a flow rejection rate approximately 20 percent higher than the optimal rate (achieved using an optimization algorithm) for low network workloads, and around 10 percent higher for high workloads.

2.3 Deep Reinforcement Learning

2.3.1 AI Overview

Artificial Intelligence refers to the ability of computer systems to perform tasks that typically require human-like intelligence. These tasks can resemble reading through Natural Language Processing, or other experiences such as pattern recognition, decision making and learning from experience. For this to be possible AI algorithms and computational models process and analyze large amounts of data, with the aim of extracting meaning and understanding and making predictions or decisions based on that analysis.

Machine Learning is an area of AI which has its focus on imitating the human learning process using data and algorithms. The 4 main types are:

- Supervised Machine Learning is defined by the utilization of labeled datasets to train algorithms for data classification or outcome prediction.
- Unsupervised Machine Learning is a type of algorithm that learns patterns from unlabeled data with processes like clustering.
- Reinforcement Learning involves an agent periodically making decisions, observing the outcomes, and then autonomously adapting its strategy to attain the optimal policy.
- Deep Learning employs algorithms inspired by artificial neural networks, which mimic the structure and functioning of the human brain. Deep Learning can be used In conjunction with other types of AI to help address some limitations of each and expand their capacity.

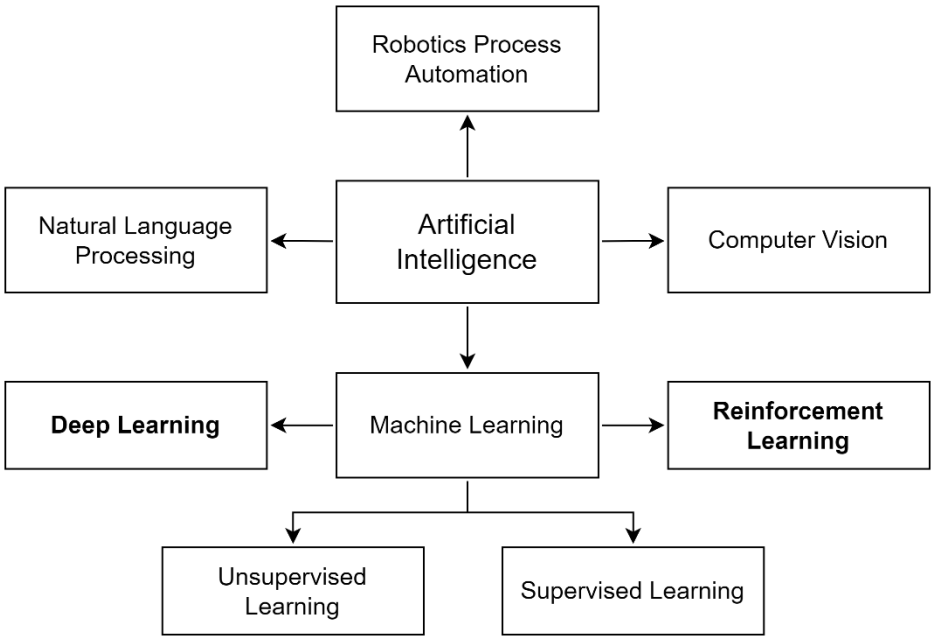

Figure 2-5 - AI Branches.

2.3.2 Reinforcement Learning

Reinforcement Learning works by having an agent interact with an environment and receiving positive and negative rewards. Subsequently the goal of the agent is to maximize the cumulative reward over a certain horizon. The rewards are used to adjust the agent's policy until it approaches an optimal policy.

Reinforcement Learning problems can be formulated as Markov Decision Process (MDP), which can be resumed as a tuple consisting of a state, an action, the transition probability from a certain state to another after an action, the immediate reward of performing an action.

In RL reaching an optimal policy with this learning process may take a long time. Deep Learning can overcome these limitations by speeding up the learning process with the help of Artificial Neural Networks. Many applications of RL, like computer vision and speech recognition, have since adopted Deep Learning techniques, this merging of technologies is called Deep Reinforcement Learning. Several studies have been made regarding the use of DRL in communications and networking [21].

2.3.2.1 Model-free vs model-based

Model-free and model-based reinforcement learning are two broad categories of approaches for learning policies in reinforcement learning.

Model-free reinforcement learning is a class of algorithms that directly learn a policy or a value function without explicitly modeling the underlying dynamics of the environment. In model-free learning, the agent interacts with the environment, receives rewards or penalties, and adjusts its policy based on the observed feedback. Model-free algorithms include Q-learning, policy gradient methods, and Deep Q-Learning.

In contrast, model-based reinforcement learning involves explicitly modeling the environment's dynamics, including the transition probabilities between states and the rewards associated with those transitions. Model-based algorithms use the known model of the environment to compute a value function or policy, which determines the agent's actions. Examples of model-based algorithms include value iteration and policy iteration.

The decision to use either model-based or model-free reinforcement learning relies on various factors, including how complex the environment is, how much data is accessible, and the amount of computing power at hand. Model-based algorithms are generally more sample-efficient and can learn faster, but they require additional computation to model the environment's dynamics. Model-free algorithms, on the other hand, are less computationally expensive and can handle complex environments without explicitly modeling them.

In practice, a hybrid approach that combines elements of both model-free and model-based algorithms is often used, leveraging the strengths of each approach to achieve better performance. Nowadays, networks have become more dynamic, making them more difficult to model.

Recent techniques such as Deep Q-Learning enable model-free algorithms that can adapt to dynamic networks much more easily [21].

2.3.2.2 Q-Learning

Q-Learning is a reinforcement learning model-free algorithm which is considered a values-based algorithm because of the way it updates the value function, which is based on the Bellman equation demonstrated in 2.1, instead of policy-based which has a primary focus on learning a policy that directly maps states to actions. It is also considered off-policy because it doesn’t consider the exploration and the agent’s actions [22].

The Q-learning agent utilizes a table of Q values for each state-action pair. Then the bellman equation is used to update the values of this table for each action taken. After an action is taken the new Q value is computed. Rewards must be defined with our purpose in mind [22].

$$New\ Q(s,a) = Q(s,a) + \alpha.[R(s,a) + \gamma.maxQ'(s',a') - Q(s,a)] \quad 2.1$$

- *Q*(s,a)* - Expected value of action,state pair
- *New Q(s,a)* - New value for certain state, action pair
- α - Learning Rate
- *R(s,a)* - Reward for the state, action pair
- *Q(s,a)* - current Q Values;
- *maxQ'(s',a')* - Maximum expected future reward
- γ - Discount Rate

Q-Learning's simplicity makes it one of the most popular and widely used reinforcement learning algorithms. However, it also has some limitations, such as:

- The amount of memory necessary to store the Q table can be impractical in cases that require large state and action spaces.
- It may converge slowly or not at all if the learning rate is too high or too low, or if the exploration rate is not decayed properly.
- The risk of overestimation bias in the case of multiple actions with similar maximum Q-values.

Some solutions have been created as an attempt to overcome some of these limitations, such as:

- Deep Q-Learning, which uses a neural network to approximate the Q function instead of a table, allows it to handle high-dimensional or continuous state and action spaces.
- Double Q-Learning, which uses two Q functions to reduce the overestimation bias by decoupling the action selection and evaluation steps.
- Dueling Q-Learning, which decomposes the Q function into two components: a state value function and an advantage function, allowing it to learn more efficiently in environments where not all actions affect the environment.

2.3.3 Deep Q-Learning

Deep reinforcement learning is a subfield of machine learning that combines reinforcement learning with deep neural networks to learn complex decision-making policies in high-dimensional state and action spaces. It aims to address the limitations of traditional reinforcement learning algorithms, which struggle with large and complex state and action spaces. Deep Q-Learning (DQN) is based on the Q-Learning approach and tries to approximate the optimal Q function using Deep Neural Networks working as shown in Figure 2-6.

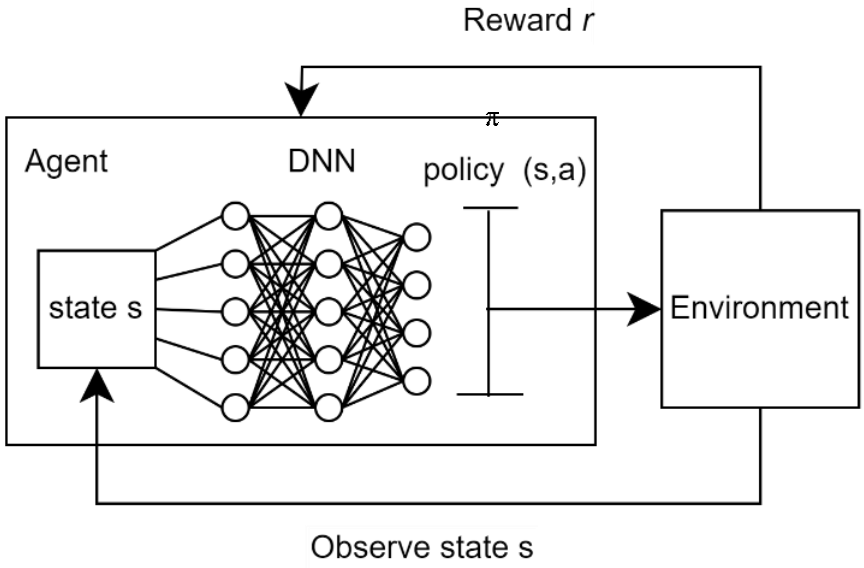

Figure 2-6 - Deep Q-Learning architecture

This solution may be prone to overestimation because the same network would evaluate and select an action. With that in mind Double Q-Learning, that uses one network for evaluation and another for estimation, was introduced and when combined with DQN improves performance by reducing overestimation [22]. Other variations were also developed like Dueling DQN and Prioritized Dueling DQN.

However, although there are many benefits to a DQN approach, naively applying Q-Learning with neural networks can lead to instability and divergence due to several reasons, such as:

- The correlation between consecutive samples in the same episode, which violates the independent and identically distributed assumption of stochastic gradient. The updates

based on correlated samples can reinforce certain errors and make learning less efficient.

- Q-Learning algorithms, including DQN, can be susceptible to overestimation bias. This bias arises because, during the learning process, the agent tends to overestimate the values of certain actions, especially in the early stages of learning. This overestimation can propagate errors and make the learning process less efficient. In the context of neural networks, the overestimation bias is amplified due to the approximation nature of the function approximators. This can lead to suboptimal policies and slower convergence to the optimal solution.

To address these issues, DQN introduces two key techniques, experience replay and target network:

- Experience replay is a method of storing and sampling transitions from a replay buffer, which breaks the temporal correlation and improves data efficiency. Each transition consists of a tuple containing the current state, the action taken, the reward received, and the next state. The replay buffer is filled with transitions collected by following an epsilon-greedy exploration policy. At each training step, a mini batch of transitions is randomly sampled from the buffer and used to update the network parameters.
- Target network is a copy of the original network that is updated less frequently (e.g., every N steps or episodes). The target network is used to generate the target Q values for training, which reduces the non-stationarity and stabilizes the learning process.

Along with these elements there are additional considerations and parameters to consider when developing a DQN application. These are some of the main topics to address when implementing a DQN algorithm:

- Choosing an appropriate network architecture and hyperparameters (e.g., learning rate, batch size, buffer size, update frequency) for the given problem domain.
- Balancing exploration and exploitation using epsilon-greedy or other exploration strategies (e.g., Boltzmann SoftMax).
- Evaluating and monitoring the performance and convergence of the algorithm using metrics such as average return, number of episodes, or learning curves.
- Extending or modifying the algorithm to handle more complex or challenging scenarios, such as continuous action spaces, partial observability, multi-agent settings, etc.

2.3.4 DRL Applied to flow scheduling in deterministic networks

DRL has been a hot topic in networking research with a surge of applications for various networking problems such as routing and scheduling happening recently. As shown in Table 2-1 at the time this dissertation started being developed only two papers had been published using DRL for flow scheduling optimization, since then over ten publications appeared offering diverse scheduling solutions using DRL. Unfortunately, there was no visibility of these works at the time of development of this dissertation.

Table 2-1 - Previous works overview

	AI/RL solutions		**DRL solutions**	
Year of publishing	< 2022	>= 2022	< 2022	>= 2022
Flow Scheduling in Deterministic Networks	5	4	2	8
Asynchronous Flow Scheduling in Deterministic Networks	2	0	0	3

Some of these research papers will be discussed further including the ones that have been released after the main period of development of this dissertation. Although there was no time to integrate these ideas into the dissertation it is still useful to discuss in order to provide greater context into how this dissertation fits in the current research landscape.

2.3.4.1 DRL applied to flow scheduling in deterministic synchronous networks

Most work done in the field of DRL, and other AI models applied to deterministic networks has been on synchronous networks that benefit from having a synchronized clock between them and typically use time-aware schedulers as explained in chapter 2.2. Some of these recent applications will be discussed along with their advancements in optimizing synchronous scheduling in the context of deterministic networks.

In [23] the authors introduce a novel approach for optimizing a Time-Aware Shaper using Deep Reinforcement Learning. In conventional TAS traffic scheduling mechanisms, guard bands are commonly employed to protect time slices dedicated to critical traffic, ensuring high-priority and essential traffic services. However, despite their role in safeguarding high-priority traffic, guard bands have several performance challenges. These include bandwidth loss, constraints on the minimum obtainable time slice length, and limitations imposed by cycle periods. Therefore, there Is a need for the development of more efficient TSN traffic scheduling schemes aimed at optimizing network utilization. The proposed DRL-based scheduling mechanism aims to address these issues by reducing the number of guard bands, thus saving more bandwidth, and freeing up additional transmission resources for Best Effort traffic. The authors conducted simulations to evaluate their algorithm's performance and found that it could efficiently find the optimal solution within an acceptable time frame.

In [24] the authors focus on addressing the routing and scheduling problem of mixed-criticality DetNet flows to ensure deterministic performance guarantees. To tackle this problem, the authors propose an approach that incorporates DRL into the Cycle Queuing and Forwarding (CQF) mechanism. While the primary objective of this paper is not to minimize end-to-end delay, the proposed approaches aim to schedule the flows with end-to-end delays close to their deadlines without excessively utilizing network resources. In other words, the main goal is to provide deterministic performance guarantees for mixed-criticality DetNet flows without compromising network efficiency. The authors assessed the DRL solution's performance by conducting simulation experiments. The outcomes Indicated its superiority over alternative benchmark techniques, such as heuristic-driven and other AI solutions. Specifically, the DRL approach can successfully schedule more flows, showcasing its efficiency and effectiveness in addressing the routing and scheduling challenges in mixed-criticality DetNet environments.

In [25] the authors propose to optimize TAS, with "DeepScheduler," an innovative and efficient flow aware TAS scheduler based on DRL. They highlight that traditional heuristic approaches for TAS often heavily rely on expert knowledge or specific assumptions, sacrificing schedulability for faster runtime by exploring only a subset of the solution space. Moreover, these methods are often tied to specific assumptions and may require redesigning if the assumptions or data distribution change slightly. In contrast, DeepScheduler automates the learning process and effectively develops scheduling policies by comprehending the intricate dependencies among data flows. It leverages Deep Reinforcement Learning to make intelligent

decisions on time slot allocation and flow selection in a centralized manner across the network. To evaluate DeepScheduler's performance, the authors conducted extensive experiments using both simulation and physical testbeds. The results showcase remarkable achievements: DeepScheduler runs over 150 times faster in simulations and more than 5 times faster on physical testbeds when compared to state-of-the-art methods. Furthermore, DeepScheduler significantly improves schedulability by 36% and 39% in the respective scenarios, outperforming traditional approaches.

2.3.4.2 DRL applied to flow scheduling in deterministic asynchronous networks

Research has been done also in the use of DRL and other AI models to optimize asynchronous deterministic networks and some of this research will be detailed further.

The authors in [26] focus their work on asynchronous TSN based Transport Networks (TNs) in 5G. They consider the use of the TSN asynchronous traffic shaper ATS at the output ports of each TSN bridge. The authors propose a novel solution for traffic prioritization in asynchronous TSN-based 5G TNs using multi-agent DRL. They introduce an offline DRL-based approach to compute long-term configurations for asynchronous TSN-based 5G TNs. The solution presented involves the mapping of 5G network slices onto IEEE 802.1Q Traffic Classes (TCs). It then allocates the TN delay budgets for these TCs across the various TN hops along predefined routes while establishing priorities for the TCs. Feasibility of a configuration is contingent upon its adherence to the end-to-end TN delay budgets set for all 5G network slices. Additionally, the solution's design can be adapted to suit different optimization objectives, whether it's maximizing the profit for TN operators or minimizing the rejection probability of 5G network slice streams. The researchers in this study focused on exploring a reward function that involves the trade-off between the priority level of a particular Traffic Class (TC) and its corresponding delay. Their main idea is that when the priority level of a TC is reduced, its delay increases. However, this change in priority may not always affect the delays of other traffic classes. This observation opens up opportunities for optimization, as adjustments in priority can lead to improved overall performance, affecting different traffic classes in distinct ways. To validate their proposal, the authors conducted a simulation-based proof-of-concept using traffic characteristics derived from common industrial 5G applications.

In [27] the authors propose a novel approach for transmission scheduling in Cognitive Internet of Things (CIoT) networks using RL techniques. The proposed approach uses RL to learn the optimal scheduling policy for CIoT networks. To evaluate the performance of the proposed approach, the authors conducted extensive simulations comparing it to several existing transmission scheduling mechanisms. The results show that the proposed approach achieves higher network throughput and lower packet delay, particularly under heavy traffic loads. The research concludes that the Q-learning-based transmission scheduling mechanism is a promising approach for CIoT networks, as it can adapt to changing network conditions and traffic patterns while maintaining high QoS levels. This research provides a comprehensive analysis of the problem addressed in the dissertation, which focuses on optimizing delay through the enhancement of transmission scheduling using reinforcement learning techniques. The author's proposed solution aims to operate Inside a network switch, where the reinforcement learning algorithm is utilized to determine the order in which packets should be transmitted. This approach requires increased computational power and intelligence at the switch level, presenting a distributed algorithm that operates within the network.

For this dissertation, the proposed solution distinguishes itself from the one in [27] by implementing the algorithm in a centralized manner at the control level of the network. By making decisions at the moment a flow enters the network, the proposed algorithm acts centrally and assigns fixed priorities to flows throughout their lifetime. While this approach necessitates a centrally managed network, it alleviates the switches from the responsibility of making such decisions.

a) LEARNET

In [28] a new approach called LEARNET is presented to tackle the online flow allocation problem using Reinforcement Learning. Performance is measured in terms of operator revenue and is compared with a baseline solution. The Forwarding plane consists of DetNet Edge Nodes Transit Nodes that implement an Urgency-Based Scheduler as proposed in [14] using the Length Rate Quotient algorithm.

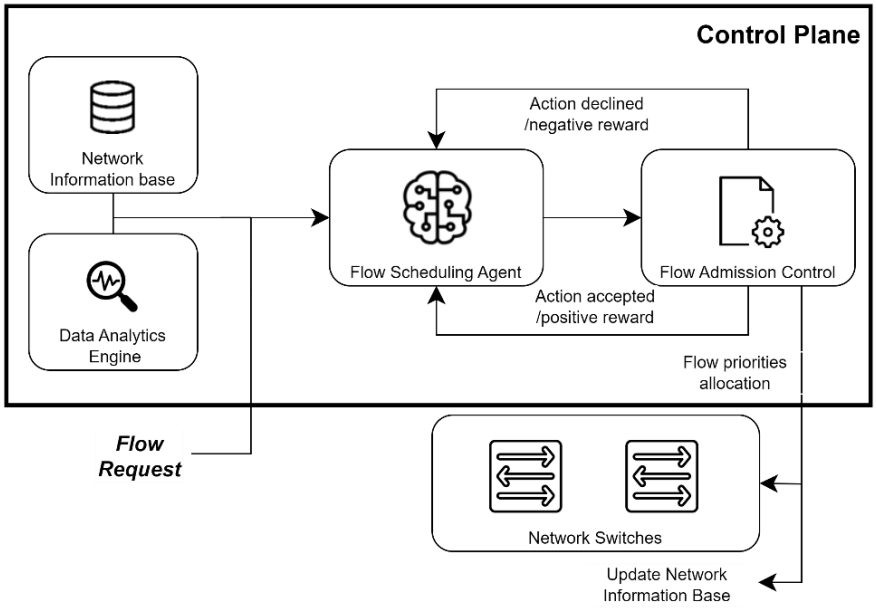

Figure 2-7 - LEARNET Architecture

The Flow Scheduling Agent (FSA) in the Control Plane needs to obtain information to perform decisions. After there is a flow request the Information about the flow is parsed and fed into the FSA, then some predictive analytics based on network Information are also fed into the FSA. The FSA will also need information about the network, such as the link capacities, current state of the shaping buffers in the Queues of the designated path and the allocated rate, allocated burstiness, minimum delay budget and maximum packet length at each priority level of every switch in the flow path. The FSA suggestion is then sent to a block called Flow admission control which based on the actions taken by the FSA checks if the flow can fulfil 5 necessary conditions:

- The end-to-end delay (E2E) experienced by a flow should not exceed its designated E2E delay budget (Df,max).
- Ongoing flows must maintain an E2E delay below their individual E2E delay budgets.
- The total data rate allocated to any link must not exceed the link's capacity.
- The accumulated burstiness allocated to any shaping buffer must remain below its designated size.
- All quality assurance and reliability rules defined by UBS, including QAR1, QAR2, and QAR3, must be adhered to

According to the result of this operation, rewards are then given and fed back to the FSA. Finally, if the flow is accepted resources will be allocated accordingly. The performance of the

LEARNET architecture using RL was compared with a baseline solution, and it managed to achieve a 45% gain in revenue for the operator [28].

2.3.5 DRL Applied to other networking applications

In [21] several applications for DRL in communications and networking are presented, that currently cover and optimize topics like rate control and network access, offloading and caching strategies, preservation of connectivity and security measures, as well as resource scheduling and traffic routing.

In [29] the authors propose the use of DRL for enabling model-free control in communications and present an effective DRL-based framework of control which aims to tackle the fundamental networking problem of traffic engineering (TE), more specifically the problem of seeking a solution that is able to specify the traffic load that goes through each path that is available in the network. Simulation results indicate that DRL-TE consistently enhances total utility and reduces end-to-end delay when compared to various commonly employed baseline techniques, all the while delivering comparable or superior throughput.

In [30] the authors propose PnP-DRL as a way to tackle the challenge of deploying DRL into real systems, which is an offline-trained DRL solution uses batch reinforcement learning through pre-collected data to achieve the best possible policy. While Deep Reinforcement Learning has gained prominence as an approach for addressing numerous networking challenges driven by experience, the practical implementation of these DRL agents encounters genuine obstacles. In this study, the authors initiated their analysis by demonstrating the potential instability and hazards that online DRL solutions may pose to operational systems. The authors proposed the utilization of and technique called Batch-Constrained deep Q-learning (BCQ), to train a plug and play deep RL agent offline. This approach allows for seamless integration and immediate effectiveness without the need for further interactions with the environment. To evaluate the effectiveness of their proposed solution, the authors implemented and assessed it in the context of Dynamic Adaptive Streaming, a prevalent experience-driven networking problem that consists of a video streaming technique that aims to dynamically adjusts the quality of video content based on the network conditions and the capabilities of the receiving device. The experimental results demonstrated that the proposed plug and play DRL Agent outperformed baseline adaptive bitrate algorithms, while on the other hand having similar performance to online solutions. By leveraging batch RL and introducing innovative

techniques, the researchers successfully developed a plug and play DRL Agent that can be readily applied in experience-driven networking scenarios. The findings of this study highlight the potential of offline-trained DRL Agents to improve networking performance without requiring real-time interactions with the environment.

In [31] the authors propose a novel approach for designing scheduling and routing algorithms for TSN using machine learning techniques. The proposed approach uses stream similarity partitioning to group traffic streams that have similar characteristics and requirements. The partitioning is performed using a clustering algorithm, which divides the streams into groups based on their similarity. The scheduling and routing algorithms are then designed based on the characteristics of each group, which allows for efficient resource allocation and avoids conflicts between streams. To train the machine learning model, the authors propose a reinforcement learning-based approach that uses a reward function to optimize the scheduling and routing decisions. The model is trained on a set of sample networks, and the learned policies are used to make real-time scheduling and routing decisions. The research presents a comprehensive analysis of the proposed approach's performance and compares it with several state-of-the-art scheduling and routing algorithms. In terms of latency, jitter, and packet loss the proposed approach was shown to outperform existing algorithms. The paper concludes that the learning-based scalable scheduling and routing co-design approach is a promising solution for TSN, as it can efficiently allocate resources and avoid conflicts between traffic streams, leading to improved network performance and reliability.

3

ENVIRONMENT SIMULATION

For the network simulation environment OMNET++ is used. OMNET++ offers a framework, GUI, library, and integrated development environment (IDE) for building different types of networks. The tool includes a wide range of models, extensions, and plugins, enabling the simulation of various networks. OMNET++ is built on the concept of models that are constructed using modules connected through gates. These modules, written in C++, utilize the simulation library provided by OMNET++. The models built using modules create the simulator, and the structure of the network is defined using the Network Description (NED) language.

The INET Framework, which is interconnected with OMNET++, provides a model library that allows users to use different Internet stack and link layer protocols. It foOverllows the same design concept as OMNET++ regarding modules, gates, and connections, and can be used to build complex networks with different network devices. The framework also provides visual support during network simulation, allowing the user to view dropped packages, path activity, routing tables, and other essential aspects. The implementation of the network simulation is achieved in OMNET++ by integrating modules and sub-modules into the network, with the modules for TSN switches, queues and traffic generators being modified and extended with extra functionality that will be further detailed In this chapter.

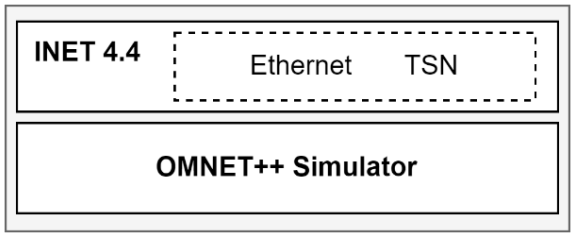

Figure 3-1 - Development environment

3.1 Network behavior

The network topology adopted in this dissertation is a standard dumbbell architecture. This configuration is comprised of two clients, each playing host to three distinct User Datagram Protocol (UDP) source applications.

This topology consists of two sub-networks connected by a bottleneck link, and has the following advantages:

- The bottleneck link between the switches represents a point of congestion and potential delay. By assigning priorities to flows and controlling their routing through the switches, it is possible to observe the impact on delay reduction and overall network performance.
- The dumbbell topology is relatively simple, which can make it easier to configure and analyze. With only two servers, three switches, and two clients, the network structure is straightforward and manageable.
- With a limited number of nodes, you have more control over the traffic patterns and flow characteristics. You can experiment with different flow rates, prioritize specific flows, and observe the effects on delay and throughput.
- While the dumbbell topology is simple, it can be extended or modified to include additional switches, servers, or clients if needed. This allows for scalability and the ability to test different network configurations.

However, it's important to consider the limitations of the dumbbell topology:

- The dumbbell topology may not fully capture the complexity of real-world network scenarios. In order to study more intricate network structures or larger-scale deployments, alternative topologies should be considered.
- With only two switches and a small number of nodes, the interactions between flows may be limited.

The corresponding flow configurations for the UDP applications employed in the simulation have been tabulated in Table 5-1. This table provides a comprehensive overview of the different flow parameters considered in the experiment, including packet sizes, inter-arrival times, and flow durations. The configuration of these parameters can significantly impact the network performance and, therefore, is of vital importance in the experiment.

The packet route for each of the UDP applications is mapped out and represented in Figure 3-2. This figure provides a visual representation of the network topology used in the simulation, including the flow of data between the different network components.

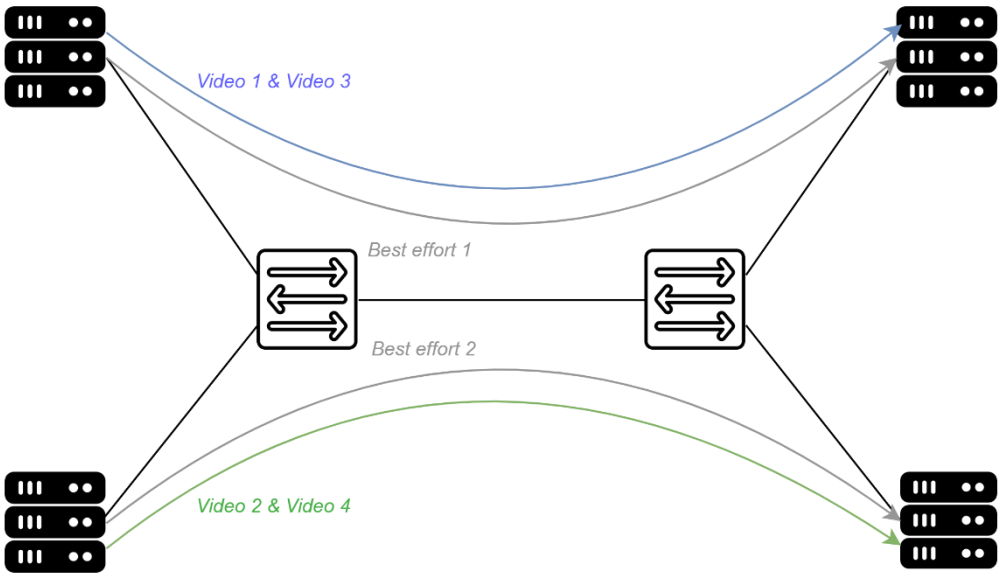

Figure 3-2 - Network topology using a dumbbell architecture

At the other end of the network topology, two servers were deployed, each employing a specialized UDP sink application. These applications were responsible for receiving the data packets transmitted from the clients and processing them.

The UDP sink application, unlike the source application, is a passive receiver that does not send any packets to the network. Its primary function is to receive and analyze the data packets sent by the clients and provide feedback on the network's performance. The data packets received by the UDP sink application are typically measured and recorded for performance analysis, such as packet loss, delay, and throughput. Here, the names Video 1, Video 2 etc. were the ones already in use by the library of simulations in INET and are meant to represent any application with a delay requirement, not just video-conferencing or other video applications.

3.2 Network configuration

To properly configure a network, it is necessary to specify a set of parameters that define the expected behavior of each model within the network. These parameters can be seen as a set of rules that govern the interactions between different components of the network.

3.2.1 Client module

Table 3-1 provides a representation of the parametrization of the UDP client applications used in the experiment and depict the different parameters that were considered while configuring the client applications.

Three applications were configured for the first client, and they shared the same message length. Despite being directed to different ports, both applications targeted the first server.

Message length refers to the size or length of the data packets or messages transmitted within the network. It represents the amount of information contained in each packet. Messaging length can affect various aspects of the network, including transmission time, bandwidth utilization, and network congestion. Longer message lengths typically require more transmission time and can consume more network resources, potentially leading to increased delays and reduced network performance.

Burst duration refers to the duration of an iteration or burst of application activity within the network, which is what is referred as a flow in this dissertation. It represents the period during which a set of packets or messages are sent consecutively as part of the flow. Longer burst durations may lead to sustained high levels of traffic, increasing the likelihood of congestion and potential packet loss.

Sleep duration represents the duration of the idle period or time interval between flows of the application. It denotes the time that the application remains inactive before resuming its activity. Longer sleep durations may result in reduced network utilization during inactive periods, potentially allowing for better resource allocation and reduced congestion.

Send interval refers to the time interval between the transmission of individual packets within a flow. It determines the rate at which packets are sent or generated by the flow. Smaller send intervals lead to higher packet transmission rates, potentially increasing network congestion, while larger send intervals may result in more spaced-out packet arrivals and lower network utilization.

The delay limit represents the maximum acceptable delay or latency for the packets transmitted within the network. It signifies the threshold beyond which the delay is considered unacceptable.

It is worth noting that the assigned values for these parameters were not strictly fixed, but rather followed an exponential distribution with a mean value aligned with the assigned values.

Table 3-1 - Clients across the board config

Client	Client 1			Client 2		
Application	Best effort	Video	Video 3	Best effort 2	Video 2	Video 4
Type	UDP Basic Burst	UDP Basic Burst	UDP Basic Burst	UDP Basic Burst	UDP Basic Burst	UDP Basic Burst
Destination Address	Server1	Server1	Server1	Server2	Server2	Server2
Message Length	900B	900B	900B	900B	900B	900B
Destination Port	1000	1001	1002	1000	1001	1002
Burst Duration	5ms	4ms	4ms	5ms	4ms	4ms
Sleep Duration	3ms	2ms	2ms	3ms	2ms	2ms
Send Interval	300us	150us	150us	300us	150us	150us
Delay Limit	-	1400us	1000us	-	1000us	1550us

For the second client there were also three applications setup that share the same characteristics as the ones in the first client although with the second server being the receiver.

It is important to note here that both video and video 3 share the recipient in server 1, while video 2 and video 4 share the recipient in server 2. This is a relationship that will affect the performance of each application and that may present a challenge for the DRL agent to optimize.

3.2.2 Server module

Servers were developed with UDP Basic Burst also but by assigning the destination address field as empty it works solely as data sink and therefore it will be only receiving the packets from the client modules.

It is in the server module that the UDP Sink app evaluates the end-to-end delay of each arriving packet and compares it to the flow's delay requirements, if it is higher than the requirement the packet is subsequently dropped, although in a real-life scenario it wouldn't be

advised to drop each packet that arrives beyond the delay limit, but in the case of this simulation and the scope of the dissertation it provides an easy way to visualize the network's performance in terms of delay guarantee by looking at the amount of packets dropped. The delay value itself of each packet is also at the same time being recorded.

3.2.3 Switch module

Each switch operates with standard ethernet protocols and in each interface of the switch a UBS queueing model is present.

This queueing model is based on the UBS described in chapter 2 and attempts to mimic Its behavior. This is achieved by modifying the source code of the switch and queue modules in INET to obtain the desired structure and functionality according UBS queueing model. It is therefore a necessary and relevant development component for this dissertation and can in the future be released as support for further investigation in the topic of UBS in deterministic networks.

The queue model is composed of 4 elements as represented in Figure 3-3. The classifier takes the packets that arrive from the buffer and allocates them to the correct queue according to the specified priority for the flow, the scheduler then makes the decision on which packet to transmit next. Both the classifier and scheduler diverged in major ways to a typical priority queue and had to be manually implemented, the rest of the modules were adapted from the INET library.

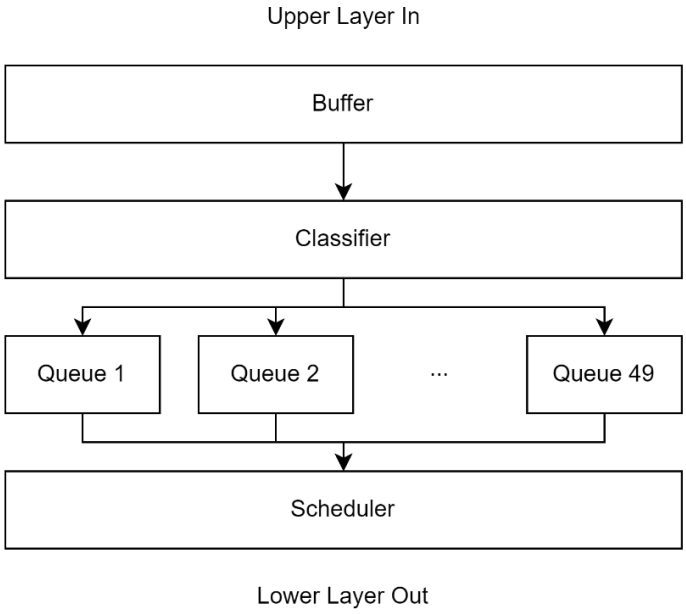

Figure 3-3 - Switch queueing model

For the classifier the first challenge, and one that is particular to this implementation, was the assignment of the priority. For our application the priority will be communicated by the DRL Agent, in a process that will be detailed in chapter 4 so it is necessary first for the classifier to be able to read the output of the DRL Agent. The classifier module then assigns the flow to the correct shaped queue.

The mechanism utilized by the classifier to do so is demonstrated in the algorithm below.

ALGORITHM 1: QUEUE CLASSIFIER

Output: *Assigned Queue for the given flow based on local and previous priorities*

Initialization of variables: *assign zero to queue, priority, previous_priority variables*

priority ← *extract priority from JSON file*

if (*we are in switch 1****)***

    if (*flow comes from client 1****)***

        queue ← ***priority*****2+1*

    else (*flow comes from client 2****)***

        queue ← ***priority*****2*

else *we are in switch 2*

    previous_priority ← *extract previous switch priority from JSON file*

    queue ← ***priority*****7+****previous_priority***

return queue

As discussed previously UBS utilizes shaped and shared queues, the number of shaped queues is determined by the number of possible priorities in the previous and current switch (e.g., if a switch as N priorities and its previous switch has M priorities, the required number of shaped queues for the switch is N x M).

For our case, since we utilize 7 priority levels at all times, for switch 2 the number of shaped queues (7 x 7) will be 49 to account for all combinations. Switch 1 operates differently since it is connected to the clients that don't operate with distinct priorities, hence the number of shaped queues (2 x 7) necessary to cover all combinations being 14.

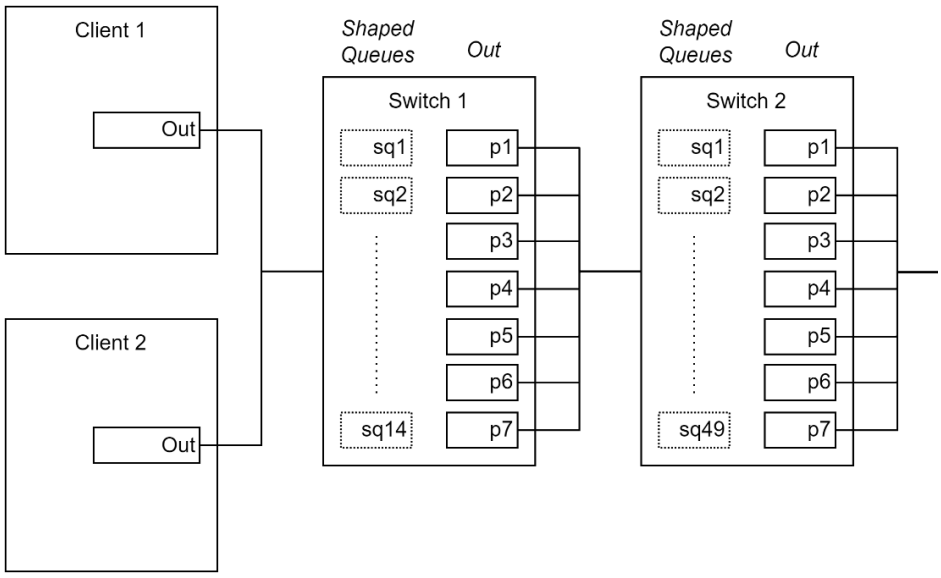

Figure 3-4 - Switch queueing model

After assigning the correct shaped queue the scheduler then transmits from them. As explained in [14], shared queues are often mentioned as pseudo queues. The term "pseudo queue" is used when assuming that the scheduler can efficiently transmit data directly from the shaped queues. In this scenario, the scheduler determines which packet to transmit next by comparing the priority levels and waiting times of the packets at the front of the queue. The operation of the scheduler is described by the algorithm below through a linear iteration performed over all queues and head of queue packets in order to identify the urgency of each packet that is then the ultimate decider of the next packet to transmit.

ALGORITHM 2: QUEUE SCHEDULER

```
    Output: Index of next packet to transmit
1   Initialization of variables: assign -1 to index and baseline, assign 0 to i
2   for (i, all queues, i++)
3       if (switch 1)
4           priority ← i/2
5       else (switch 2)
6           priority ← i/7
7       waiting_time ← compute waiting time based on current time and packet arrival time
8       urgency ← priority*waiting_time
9       if (urgency > baseline)
10          Baseline ← urgency
11          index = i
12  return index
```

The algorithm described starts by understanding the priority of the queue based on its index and then uses the priority and the waiting time of the packet to establish the urgency of the packet.

3.3 Network visibility

To effectively monitor and maintain the network and also enable the DRL Agent's actions, it is important to have a comprehensive understanding of the network's performance and state. This involves collecting and analyzing data from various sources to gain insight into how the

network is functioning. To achieve this, a variety of monitoring and visibility mechanisms can be employed, including the creation of specialized files that serve different purposes.

To this end, four distinct files were created to capture the network's state and performance. Network events and network state are recorded in two files:

- NetworkState.json records the buffer activity in switches and the state of active flows in the network, providing insight into the network's overall activity and traffic patterns.
- FlowCharacteristics.json, on the other hand, records the characteristics of new incoming flows, including end-to-end delay budgets, allowing for a more granular understanding of how individual flows are impacting the network.

Network performance is recorded in two additional files:

- FlowsTiming.json records the start and end dates of any given flow, allowing for the analysis of flow duration and the identification of potential issues or bottlenecks in the network.
- DroppedPackets.csv records every packet dropped during the simulation, along with a timestamp, providing critical data for analyzing packet loss and identifying potential sources of congestion or other network issues.

Taken together, these specialized files provide a comprehensive view of the network's state and performance.

In Figure 3-5 it is possible to understand from which elements of the network each file collects information.

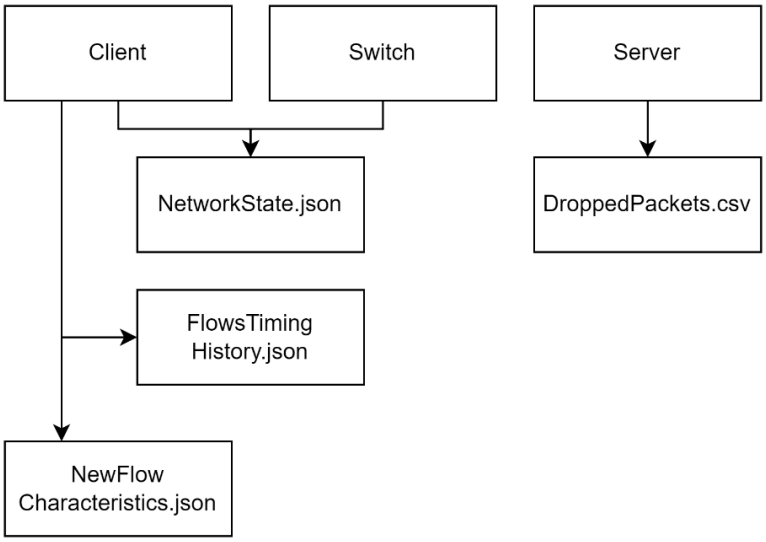

Figure 3-5 - Files representing the network simulation

3.3.1 Network state

To ensure comprehensive visibility over the state of a network, it is important to collect and analyze a wide range of sources. One key data source is information on the state of queues within the network, which can provide critical insights into network traffic patterns, performance, and potential issues.

To achieve this level of visibility, each switch within the network is continuously updating the JSON file with information about the length of its queues. This data is collected in real-time and will allow the DRL Agent to make better informed decisions.

It is important to note here that only specific queues were designated to be included in the network state. These queues are only from switch 1 since it's the only choke point in the network. From switch 1 only a subset of the queues is relevant to represent for the DRL agent since the actions of the agent were limited to a set of priorities smaller than 7 with the goal of simplifying the learning process and actions of the agent as will be explained in the next chapter.

```
{
    "switch1.eth[0].macLayer.queue.queue[0]": 0,
    "switch1.eth[1].macLayer.queue.queue[0]": 0,
    "switch1.eth[2].macLayer.queue.queue[0]": 0,
    "switch1.eth[2].macLayer.queue.queue[12]": 69,
    "switch1.eth[2].macLayer.queue.queue[13]": 53,
    "switch1.eth[2].macLayer.queue.queue[1]": 3,
    "switch1.eth[2].macLayer.queue.queue[2]": 0,
    "switch1.eth[2].macLayer.queue.queue[3]": 20,
    "switch1.eth[2].macLayer.queue.queue[4]": 0,
    "switch1.eth[2].macLayer.queue.queue[5]": 10,
    "switch1.eth[2].macLayer.queue.queue[6]": 0,
    "switch1.eth[2].macLayer.queue.queue[7]": 0,
    "switch1.eth[2].macLayer.queue.queue[8]": 127,
    "best effort 2|activation status": true,
    "best effort 2|lifetime": 7385,
    "best effort 2|nextBurst": 1.035212523883,
    "best effort 2|nextSleep": 1.034367951816,
    "best effort 2|priority": 6,
    "best effort|activation status": true,
    "best effort|lifetime": 7530,
    "best effort|nextBurst": 1.035706916259,
    "best effort|nextSleep": 1.033873743575,
    "best effort|priority": 6,
    "simtime": 1.029793650755,
    "video 2|activation status": true,
    "video 2|lifetime": 4759,
    "video 2|nextBurst": 1.032280834192,
    "video 2|nextSleep": 1.031390535513,
    "video 2|priority": 0,
    "video 3|activation status": true,
    "video 3|lifetime": -4187,
    "video 3|nextBurst": 1.036464116566,
    "video 3|nextSleep": 1.036179865543,
    "video 3|priority": 0,
    "video 4|activation status": true,
    "video 4|lifetime": -5083,
    "video 4|nextBurst": 1.030521341262,
    "video 4|nextSleep": 1.03036118355,
    "video 4|priority": 4,
    "video|activation status": true,
    "video|lifetime": 7297,
    "video|nextBurst": 1.03275229537,
    "video|nextSleep": 1.032282983671,
    "video|priority": 1
}
```


Figure 3-6 - Representation of NetworkState.json file

In addition to monitoring queue state, it is also important to track the behavior of active flows within the network. To achieve this, client modules are continuously updating the same JSON file with information about all active flows that are transmitting in the network as well as their expected lifetime.

Taken together, these data sources provide a comprehensive view of the state of the network.

This information will be the basis of what the DRL Agent has to work with in order to make a correct assignment of priorities.

3.3.2 Flow characteristics

When a new flow begins in the client module its name is recorded and written to a JSON file along with the delay limit associated with the flow.

The modification of this file to account for the creation of a new flow will be the trigger for the DRL Agent to act.


```
{
"video-1": "0.001"
}
```


Figure 3-7 - Representation of FlowCharacteristics.json file

3.3.3 Flows timing history

In the client module the start and end time for each flow are recorded and written to a JSON file. Each flow is identified by the application/class it belongs to (e.g., video 2) followed by a unique identifier (e.g., (1) meaning that it is the first flow of the type of video 2).


```
"video 2-0": {
    "end time": 191640419,
    "start time": 191627603
},
"video 2-1": {
    "end time": 163620223,
    "start time": 162842950
},
```


Figure 3-8 - Representation of FlowsTiming.json file

It is important to note that to establish a synchronized time reference between the network simulation in OMNET++ and the DRL Agent, a unified time format is adopted. The chosen format represents time ranging from hours to nanoseconds (e.g., Above for video 2-0

end time we see 191640419 which means 19 hours 16 minutes 40 seconds etc.), allowing for precise temporal alignment and coordination between the two systems. By utilizing this shared time format, both the OMNET++ simulation and the DRL Agent can seamlessly exchange time-related information, enabling accurate synchronization and facilitating consistent analysis and evaluation of network performance.

3.3.4 Dropped packets

In the server module when a new packet arrives and is processed a comparison is made to determine whether the packet's delay exceeds the threshold value defined for the application. If the delay is found to be above this threshold, the packet is dropped and recorded to the DroppedPackets.csv file, therefore, dropped packets are a useful and easy way of measuring the number of packets that fail to achieve a satisfiable end-to-end delay.

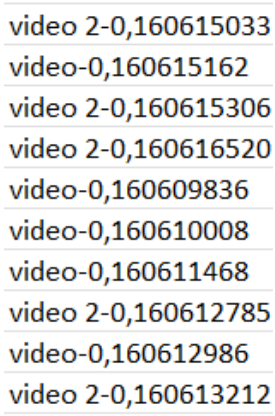

Figure 3-9 - Representation of DroppedPackets.csv

Reading this csv file, it is possible to see each entry as the flow of the dropped packet (e.g., video 2-0) and the time at which the packet was dropped (e.g., 160615033). This collection of dropped packets with the corresponding time of drop will be the basis for reward calculation in the DRL algorithm and will also be the ultimate parameter of success when evaluating the networks' performance.

3.4 Network simulation challenges

Implementing all the mechanisms discussed presented some challenges, mainly setting up the simulation according to the needs of visibility for the interaction between DRL Agent and simulation to function, but also in the configuration of each module to represent an

accurate depiction of the networking scenario we are trying to simulate. Also, the implementation of an Urgency based scheduler presented a challenge of its own.

All these changes required a big learning curve of the functioning of OMNET++ and the interactions between the different modules of INET so that they could be adapted to fit specific requirements and needs. This represented the bulk of the work in this dissertation, despite the focus being on the performance of the DRL optimizations, it was first necessary to create an environment suited for testing the aspects relevant to the problem statement.

4

DRL AGENT

With the network simulated and full visibility achieved, the next step is to implement the DRL Agent. This agent is designed to observe the state of the network, and to allocate priorities along all the switches in the network for a given flow when it arrives in the network.

In this chapter, the functioning of the DRL Agent will be explained as well as the way it interacts with the environment followed by an explanation of the rationale behind each decision when designing the neural network and the agent itself.

4.1 DRL Agent behavior

The main goal of the DRL Agent is to, when triggered by the arrival of a new flow in the network, analyze the current network state and decide on a set of priorities to allocate to the flow across the switches. To extrapolate a reward the agent then checks for the number of dropped packets in the network within the flow's lifetime and that becomes the metric of success and reward.

Along the way this approach could yield results, but unfortunately the interaction between the network simulation and the DRL Agent requires additional computational resources and time due to the reading and writing from the files that eventually build up and don't allow for the simulation to run at full speed through lengthy periods of time.

To address this implementation a batch-DRL approach is taken to training. A decision was made to simulate the network without any interaction with the Agent, collecting data on flows behavior and response to different priority assignments. This collection of data then provided the basis for training the agent offline. The agent is then ready to be deployed in the network online after being trained.

4.1.1 Offline training

To make possible the offline training of the agent a batch DRL approach was used. This is a variant of traditional DRL algorithms that addresses the challenges of learning from a fixed dataset or batch of experiences rather than interacting with an environment in real-time. A training dataset was developed by simulating the network separately and assigning random priorities observing its effect on flow performance. This training dataset is then populated with an entry for each flow and processed as a Pandas DataFrame obtained from the csv file. The illustrative structure of the DataFrame Is demonstrated in Table 4-1.

With the information on this DataFrame, it is possible to iterate each entry and assign each to a step in the learning process of the DRL Agent, thus making offline learning possible.

Table 4-1 - Dataframe structure

Flow Type	Flow ID	Priority	State	Next State	Reward
Video 2	Video 2-2	3	[1,14,...,1]	[5,23,...,0]	0.108
Video	Video-2	2	[5,14,...,1]	[12,18,...,0]	0.638
Video	Video-3	0	[3,1,...,0]	[9,1,...,1]	0.235
Video 2	Video 2-3	1	[9,4,...,1]	[1,6,...,1]	0.98

In this implementation, the network state comprises the elements extracted from the NetworkState.json file, as discussed in chapter 3.3.1. These elements collectively form a state representation of size 43, which serves as the input to the agent.

After the agent takes a random action and assigns priorities to the flow, the subsequent state captures the same vector of information as the previous state but with the updated priorities represented.

The agent's actions involve assigning priorities to flows as they arrive in the network. To simplify the action space and enhance the agent's decision-making process, the priority assignment options are limited to 0,1,2 or 3 although the network supports 7 priority levels.

After the simulation is terminated and the actions and results are recorded, a Jupyter notebook is employed to iterate over the data and take a learning step with the agent for each entry. This process is represented in Figure 4-1.

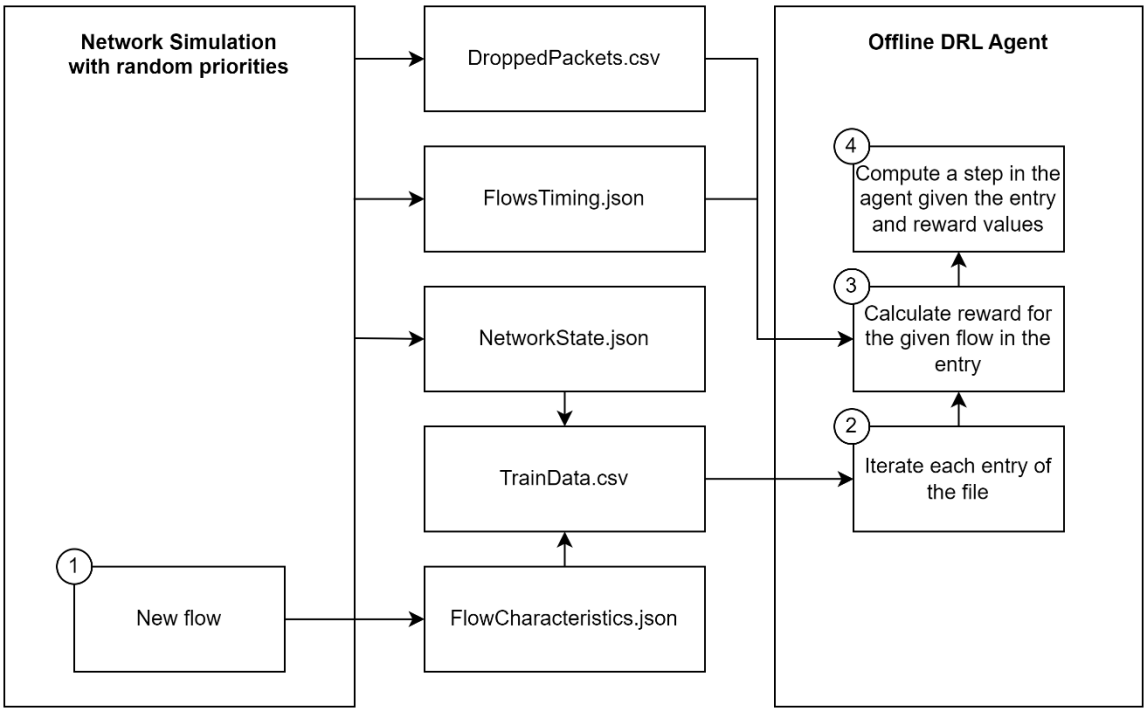

Figure 4-1 - Offline training methodology

This approach allows us to be effective with the limited computational resources and time that was available for this research. In the Figure 4-1 the process of offline training is represented.

4.1.2 Online deployment

For this application's online deployment the DRL Agent is triggered every time a deterministic flow reaches the network, when an incoming new flow arrives to the network the DRL Agent extracts the current state of the network, which is then used to compute the action, a set of priorities, that is applied directly to the network through the ModelResponse.json file, a process that is illustrated In Figure 4-2.

The properties defined to begin the implementation of this DRL Agent are presented in This approach allows us to be effective with the limited computational resources and time that was available for this research. In the Figure 4-1 the process of offline training is represented.

Table 4-2 - Step DRL Implementation properties

Property	Solution
Trigger	New flow starts transmission
State size	43 (from NetworkState.json file)
State	• Buffers state • Current simulation time • Active flows currently in network with: 1. Remaining lifetime 2. Activation status 3. Lifetime 4. Next Burst 5. Next Sleep 6. Current priority
Action size	4
Action	For the given flow a priority attributed in the range of 0 to 3

The process of online deployment is represented in Figure 4-2 where it is possible to see that although the agent was previously trained offline it is still learning while interacting with the network.

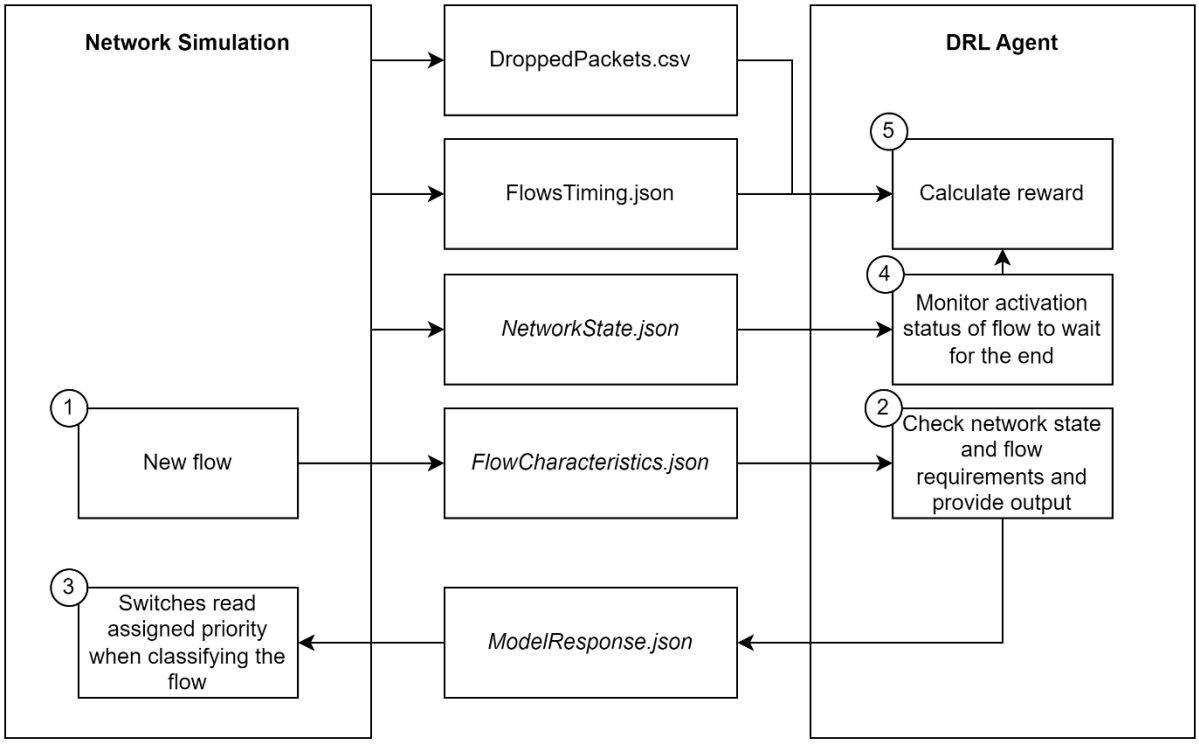

Figure 4-2 - Online agent-simulation communication

4.1.3 Reward function

The reward function is a critical aspect of DRL applications, as it influences the agent's behavior and learning process. In this study, the reward value for each flow is determined based on the number of dropped packets, which serves as a measure of network performance. The algorithm used to calculate the reward is designed to yield a value close to 1 if the number of dropped packets is low and closer to 0 if the number of dropped packets is high. To ensure a more balanced distribution of reward values, a correction factor of 0.2 is applied, making it suitable for the specific network values being simulated.

ALGORITHM 3: REWARD FUNCTION

Output: *Reward value*

Input: *flow name; FlowsTiming.json; DroppedPackets.csv*

1 ***start_time, end_time ← FlowsTiming.json***

2 ***num_dropped_packets← DroppedPackets.csv(start_time, end_time)***

3 ***flow_reward←1/(1+0.2*num_dropped_packets)***

4 ***return reward***

4.2 DRL Agent Implementation

The Agent's functionalities and mechanisms of interaction with the simulation have already been described, however, supporting this behavior and interaction with the environment exists the neural network capable of making decisions and learning through each step. Therefore, the Neural Network and its supporting functions had to be designed according to the specific needs of the application.

The Agent was implemented in python using PyTorch, a powerful deep learning framework that offers a range of features and capabilities, including an API for python that can be used to implement the network layers, optimizer, and activation functions.

To design the neural network and the supporting capabilities, it was first necessary to consider the different components and make key decisions on how to implement them, the components addressed are:

- Neural network.
- Replay buffer.
- Target network.
- Optimizer.

With the main components described it was then necessary to give the parameters of the neural network the best possible values according to the application's functionalities.

4.2.1 Components

4.2.1.1 Neural network

When implementing a DRL algorithm the choice of neural network architecture plays a crucial role in enabling effective learning and decision-making. The main architecture utilized in DRL is the Q-network, which is a specific type of neural network known as a feedforward neural network or multi-layer perceptron. The Q-network consists of multiple layers of interconnected nodes, allowing it to capture and process the intricate relationships between input states and their corresponding action values.

4.2.1.2 Replay buffer

The agent learns by keeping a memory of its experiences in a replay buffer. Each experience includes the current situation, the action it chose, the reward it received, the next situation it transitioned to, and whether the episode ended. By randomly picking experiences from this

memory, the agent can learn from a variety of situations and avoid being influenced too much by consecutive experiences. This helps the agent learn more effectively and make better decisions.

4.2.1.3 Target network

To improve the learning process's stability and effectiveness, the agent uses a target network. It keeps two versions of the neural network: the local network and the target network. The local network is used to make predictions and is updated regularly, while the target network is used to estimate the target values and is updated more slowly. The agent applies a technique called soft update to gradually update the target network by blending its parameters with those of the local network. This helps the agent make better predictions and improve its learning over time.

4.2.1.4 Optimizer

An optimizer is a key component in training neural networks that adjusts the network's parameters to minimize the error or loss between the predicted outputs and the true targets. It determines how the network learns from the training data by optimizing the weights and biases of the network's layers. The choice of optimizer plays a crucial role in the training process. In this case, an Adam optimizer was used. Adam stands for Adaptive Moment Estimation, and it is a popular optimization algorithm known for its efficiency and effectiveness in training neural networks. It aims to adaptively adjust the learning rate for each parameter. This adaptive learning rate allows the optimizer to converge faster and handle different types of data and network architectures effectively.

4.2.2 Parameters

All the components in this architecture must be parameterized to ensure the right behavior, it's essential to make thoughtful choices to achieve the best performance. Finding the right balance between the network's ability to learn complex patterns and its generalization to new data is crucial. Hyperparameters, like the learning rate, gamma, batch size, buffer size, and update frequency, also impact the network's training process and need to be carefully tuned for optimal results.

Table 4-3 - Agent parameters

Number of layers	4
Nodes in input layer	43
Nodes in output layer	4
Nodes in 1st hidden layer	23
Nodes in 2nd hidden layer	13
Buffer size	10000
Batch size	8
Gamma	0.2
Tau	0.01
Learning rate	0.01
Update frequency	1
Epsilon	0.3

Selecting the optimal values for these parameters usually requires experimentation and trying different combinations. It's important to consider the complexity of the problem, the available computing power, and learn from practical observations to make informed choices.

4.2.2.1 Neural network size

Designing the neural network itself requires a decision on how many layers to include and how many nodes to include in each layer. To determine the right number of layers for the neural network, it is necessary to evaluate the complexity of the problem. Deep neural networks with many layers can capture intricate patterns in complex datasets, but they might overfit if there isn't enough data. Alternatively, networks with fewer layers might be better suited for simpler problems or when data is limited. Striking the right balance between complexity and data availability is key to designing an effective neural network.

According to established principles in designing neural networks design [32], the number of hidden layers in a neural network plays a critical role in its ability to capture complex patterns. When there are no hidden layers, the network can only represent simple relationships between inputs, which limits its capability to handle more intricate data patterns. By introducing a single hidden layer, the network becomes capable of approximating functions that involve continuous mappings between different spaces. With the addition of two hidden layers, the network gains the capacity to accurately represent complex decision boundaries, as long

as appropriate activation functions are employed. A neural network with two hidden layers can approximate any smooth mapping with arbitrary precision.

For this dissertation, keeping in mind that the amount of available data is limited due to time constraints for simulating the environment, four layers were used, which means that apart from the mandatory input and output layers there were two hidden layers included.

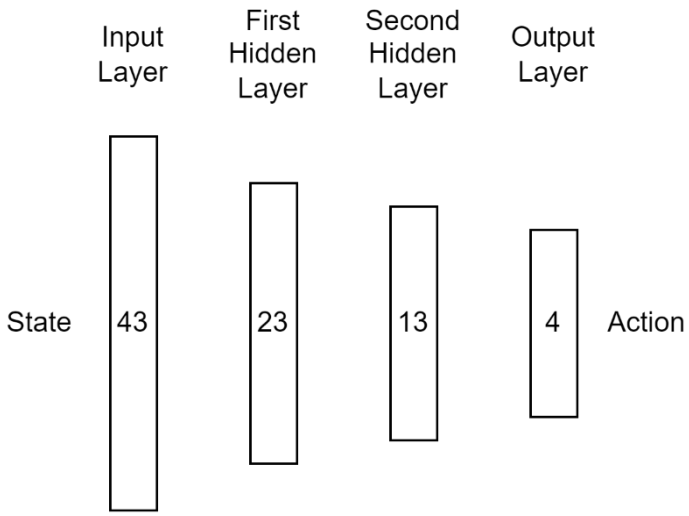

Figure 4-3 - Neural Network layers

Another important aspect to consider is the number of neurons in the hidden layers of the network. Similar to choosing the number of layers, if there are too few neurons, the network may struggle to capture complex patterns in the data. On the other hand, if there are too many neurons, the network may end up memorizing the training data instead of understanding the underlying patterns. Finding the right balance often requires experimentation and fine-tuning.

The sizes of the input and output layers are determined by the characteristics of the data and the specific task. The input layer size corresponds to the number of features or variables in the state of the environment being used, while the output layer size depends on the desired format of the output, which is the action taken. For this dissertation, given the state size, the input layer has a size of 43 and given the action size, the output layer has a size of 4.

There are some commonly recommended guidelines [33] to determine the suitable number of neurons in the hidden layers of a neural network. These guidelines include:

1. Ensuring that the number of hidden neurons falls within the range between the size of the input layer and the size of the output layer, on average.

2. Calculating the number of hidden neurons as 2/3 of the size of the input layer, plus the size of the output layer.
3. Opting for the number of hidden neurons equal to the square root of (input layer nodes * output layer nodes).
4. Keeping the number of hidden neurons less than twice the size of the input layer.
5. Gradually reducing the number of hidden neurons in subsequent layers to facilitate pattern and feature extraction and enhance target class identification.

With these guidelines in mind and after some experimentation it was decided to use guideline 1 for the first hidden layer to give us a size of 23 nodes and guideline number 3 for the second hidden layer giving a size of 13, this also respects guideline 4 and 5 by keeping a decreasing progression.

It is worth mentioning that pruning is another technique, not discussed in this dissertation, that aims to optimize the number of neurons in the hidden layers and improve computational and resolution performance. Pruning involves trimming unnecessary neurons during training by identifying those that have little impact on the network's performance. This can be done by examining the weights of the neurons, where weights close to zero indicate relatively less importance.

4.2.2.2 Activation function

When designing a neural network, it's important to choose the right activation function. Activation functions are mathematical functions applied to the output of each neuron in a neural network. These functions introduce non-linearity, allowing the network to model and learn complex patterns in the data. Activation functions determine the activation level or output of a neuron based on its input.

For this dissertation ReLU was used. Unlike other activation functions like sigmoid and tanh, ReLU doesn't become saturated for positive inputs, which helps gradients flow smoothly and improves training speed. These characteristics can be helpful given the fact that due to some constraints the data available to our agent won´t be as much as it could be.

4.2.2.3 Learning rate

The learning rate controls how big of a step the network takes when adjusting its weights during learning. A higher learning rate can make the network learn faster, but it runs the risk of overshooting the best solution. On the other hand, a lower learning rate may require more training time, but it ensures a more stable and precise learning process.

For this dissertation the learning rate of 0.01 was chosen based on several factors. Firstly, it falls within a typical range for learning rates in many deep learning applications. This moderate learning rate strikes a balance between fast convergence and avoiding overshooting the optimal solution.

4.2.2.4 Batch size

The batch size determines how many examples are processed in each training step. Increasing the batch size can accelerate the training process, although at the potential cost of higher memory usage. On the other hand, opting for a smaller batch size results in a gradient estimate with more noise.

For this dissertation a batch size of 8 was used. Using a smaller batch size can reduce the memory requirements and computational load compared to a larger batch size. Due to the limited computational resources, it is advantageous to use 8 as the batch size.

4.2.2.5 Buffer size

The buffer size, commonly used in experience replay, determines how many past experiences are stored for training. A larger buffer size allows for a wider range of experiences, which helps stabilize the learning process.

For this dissertation a buffer size of 10000 is more than enough to accommodate all experiences.

4.2.2.6 Gamma

Gamma is the discount factor that determines the relative importance of future rewards compared to immediate rewards. It affects how much the agent values long-term rewards when making decisions. A higher gamma value, closer to one, indicates that the agent values future rewards more, while a lower gamma value, closer to zero, means the agent prioritizes immediate rewards.

For this dissertation there are specificities that must be considered when choosing the gamma value. One factor to keep in mind is that the next state obtained from taking an action in the environment is not a certain depiction of future rewards, this means that future rewards in the environment are uncertain and not to be relied upon, with this in mind a gamma value of 0.2 was chosen to direct the focus to the immediate rewards, which can be predictable and much better optimized.

4.2.2.7 Tau

The tau parameter determines the extent to which the target network's parameters are updated towards the online network's parameters. A smaller tau value results in slower updates, providing more stability to the learning process. Conversely, a larger tau value leads to faster updates, allowing the target network to adapt more quickly to changes in the online network.

For this dissertation a tau value of 0.01 was used for the target network parameters to be updated at a moderately fast rate. It indicates that 1% of the local network parameters are incorporated into the target network at each update step. It allows the target network to adapt more quickly to the changes in the local network. This faster update rate may result in quicker convergence and learning progress. However, it also introduces more variability and potential instability, especially if the learning process is not carefully controlled.

4.2.2.8 Update frequency

The update frequency determines how often the network's parameters are adjusted during training. A higher update frequency means that the parameters are updated more frequently, which can lead to faster adjustments but may also introduce more variability. On the other hand, a lower update frequency means that the parameters are updated less often, which may result in slower convergence but can provide more stable updates. For this dissertation we use an update frequency value of 1, which means that after each reward is obtained, the network parameters are updated.

4.2.2.9 Epsilon

In reinforcement learning, exploration means trying out new actions and states to learn more about the environment, while exploitation involves using the knowledge gained so far to make the best decisions. Finding the right balance between exploration and exploitation is key to achieving optimal performance. At the beginning, it's important to explore more to discover new and potentially better actions.

For this dissertation, despite the agent already undergoing offline training, a fixed epsilon of 0.3 was used for the online deployment since the environment is dynamic, it may be beneficial to maintain a certain level of exploration throughout deployment and ensure the agent keeps exploring new possibilities and learning even as it exploits its learned knowledge.

4.3 DRL Agent Implementation challenges

4.3.1 Sync between Agent and simulation

The communication process between the agent and the simulation requires an intricate mechanism of reading and writing to files, it is important to keep the files from being corrupted and to ensure that the agent is not running alone.

To ensure this, the agent was programmed to await a signal from the simulation indicating that the simulation has started. From then the agent begins running alongside the OMNET++ simulation. This signaling mechanism was Implemented by adapting OMNET++ simulation source code.

4.3.2 File communication

The method for communication between the DRL Agent and the simulation was detailed in chapter 3.3, explaining the different files that were created and updated live with the simulation in order to give constant visibility to the DRL Agent. Also, the DRL Agent's actions are, as explained, communicated through a file.

An example of this file interaction for the step DRL architecture implementation is Illustrated in Figure 4-2. At the onset of a new flow in the network, the flow is logged in the FlowCharacteristics.json file, triggering the activation of the DRL Agent. The DRL Agent then reads the current network state from the NetworkState.json file and generates a response.

The constant interaction needed between simulation and agent requires that these files are constantly being opened and closed to be written or read. An obvious problem arises from the fact that there is the possibility of the two programs trying to access the file at the same time, thus leaving one of them with a corrupted version or simply without being able to open the file.

To resolve this issue a file lock system was implemented, where, as described in Figure 4-4, before a program tries to read or write to a file it first checks that it exists and its not

locked, then, if its unlocked it immediately locks it and performs the necessary operations, followed by the unlocking for it to be used by the other program.

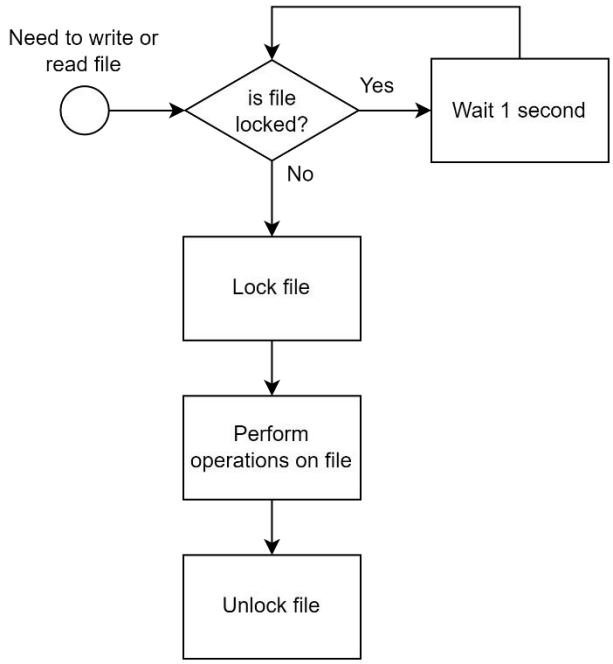

Figure 4-4 - File locking mechanism

The locking is done by creating a .lock file that serves as an Indicator to all programs in the system that the designated file is locked.

5

SIMULATION AND RESULTS

Traditionally, asynchronous schedulers have relied on traffic prioritization (fixed priorities) [18], with that in mind the network was first simulated without any interaction from the DRL Agent and optimally assigned fixed priorities based on the delay requirements of each flow. It was then simulated while interacting with the DRL Agent in order to understand if the DRL assigned priorities could consistently beat the fixed priorities, proving that the DRL Agent was able to learn the behavior of the environment. In order to expand the comparison, simulations were also made with randomly assigned priorities, with the goal of proving that the DRL performance boost was not a product of simply assigning diverse priorities per flow.

The simulations for all the proposed solutions were done under uniform conditions with the same length of time allocated to each simulation and with the characteristics presented in Table 5-1. Table 5-1 also shows the best fixed priority assignment, achieved through a logical and trial-and-error process, considering the specific requirements of each flow to ensure optimal performance.

These simulations allowed for an accurate comparison by evaluating the ultimate success criteria, which is the number of packets in the deterministic flows that fail to achieve their designated delay limit. This success criteria is tied to the premise of this dissertation, to optimize flow scheduling for deterministic networks. Although this solution doesn't aim to guarantee end-to-end delay 100% reliability, it aims to offer an optimization of reliability that can be applied to other more holistic solutions.

Table 5-1 - Flow configurations

Flow ID	Sleep Duration (ms)	Send Interval (us)	Burst Duration (ms)	Delay Limit (us)	Message Length (B)	Best fixed priority
Best effort	3	300	5	-	1000	6
Best effort 2	3	300	5	-	1000	6
Video	2	150	4	1400	1000	2
Video 2	2	150	4	1000	1000	0
Video 3	2	150	4	1000	1000	0
Video 4	2	150	4	1550	1000	3

It is also relevant to note that in the simulation, the network parameters were semi-randomly generated based on a distribution with the mean values being the parameter value specified. This approach ensured that each simulation had slight variations in the micro changes of each parameter, with the hope of introducing realistic diversity into the experiments.

For the fixed priorities solution there were 10 simulations made that performed with a consistent level of performance. If not for the slight change of parameters for each simulation, we should expect the same exact performance every time.

For the simulations with DRL assigned priorities and randomly assigned priorities an extra 10 simulations were made for a total of 20. This was due to these solutions not being as consistent as the fixed priorities solution, particularly the random solution.

5.1 Performance overview

The first analysis of performance consists of looking at the number of dropped packets these solutions are able to achieve across the various simulations. The full detailed simulations data is presented in Appendix A.

In Figure 5-1 it is possible to observe the consistency of the fixed priorities solution with an average of 178 dropped packets per simulation. This baseline number of dropped packets is also the average for the randomly assigned priorities solution. It is also possible to observe that the random assignment of priorities leads, as expected to some good, but mostly the worst results.

The DRL solution was able to offer an optimization of the number of dropped packets with an average of 103 dropped packets for all 20 simulations, which represents a 42% decrease in dropped packets, although without the same level of consistency as the fixed priorities solution. This might be explained by a sensitivity to changes in the network conditions and parameters and the inability of the DRL model to offer the best possible response to all scenarios.

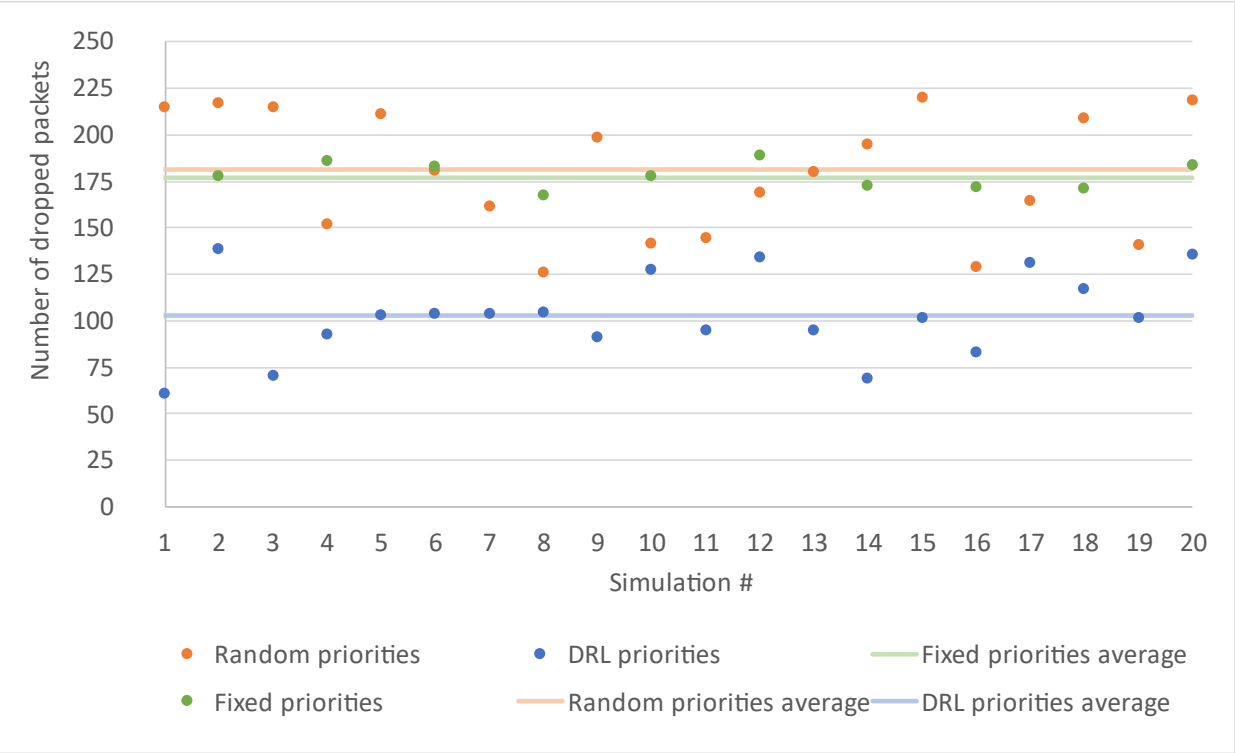

Figure 5-1 - Overall performance comparison

5.2 Per flow dropped packets performance

By looking at the number of dropped packets per flow across all simulations represented in Table 5-2 it is possible to see that the traditional fixed priorities solution is not able to optimize the number of total dropped packets because although it assigns a lower priority level to the deterministic flow with less delay requirements (video 4) it is still not able to prioritize the other flows enough in order to prevent a disparity in the number of dropped packets across all flows.

The DRL solution achieves a result closer to equity among the 4 deterministic flows that could be at the root of the optimization.

Table 5-2 - Per flow dropped packets performance comparison

	Video		Video 2		Video 3		Video 4	
	StdDev	Avg DP	StdDev	Avg DP	StdDev	Avg DP	StdDev	Avg DP
Fixed	7.39	63.4	16.62	58.1	7.41	43.2	5	13.5
Random	18.32	44.35	14.73	37.7	38.64	77.15	20.88	20.4
DRL	8.48	15.3	15.03	39.75	16.28	24.25	19.52	23.85

However, by looking at Figure 5-2, Figure 5-3 and Figure 5-4 it is possible to note that equity among all flows is not the only explanation for better performance with DRL since overall good and bad results were achieved with varying disparities between each flows performance.

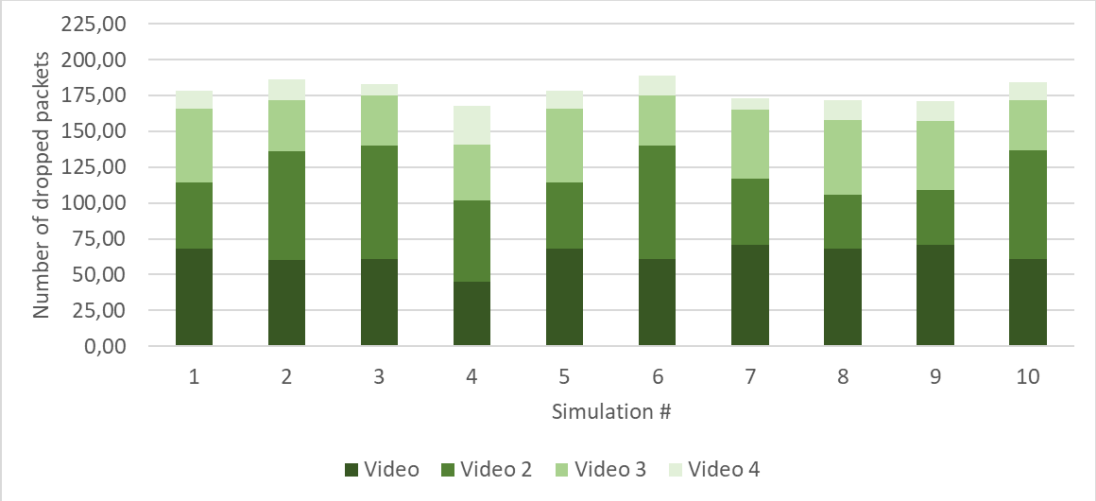

Figure 5-2 - Per flow dropped packets for fixed priorities solution

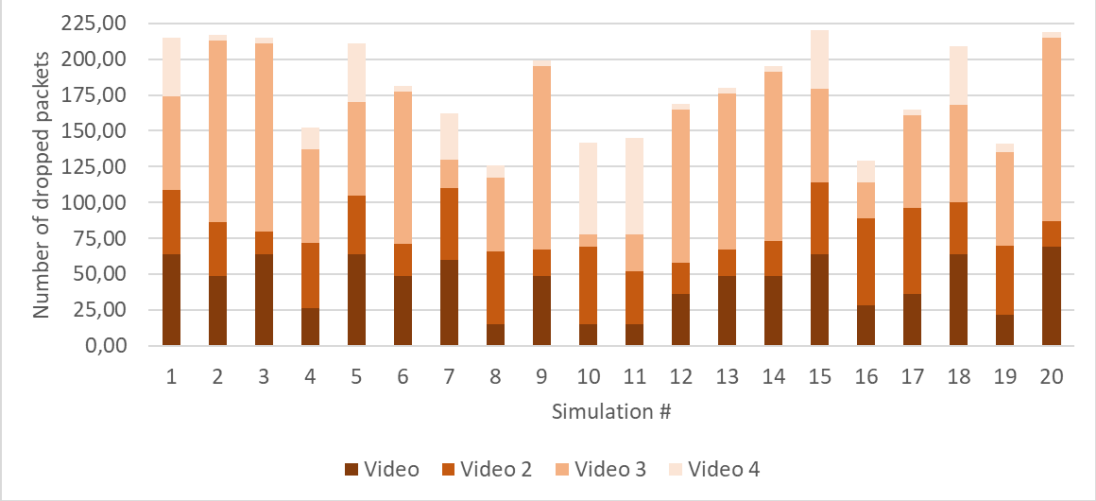

Figure 5-3 - Per flow dropped packets for random priorities solution

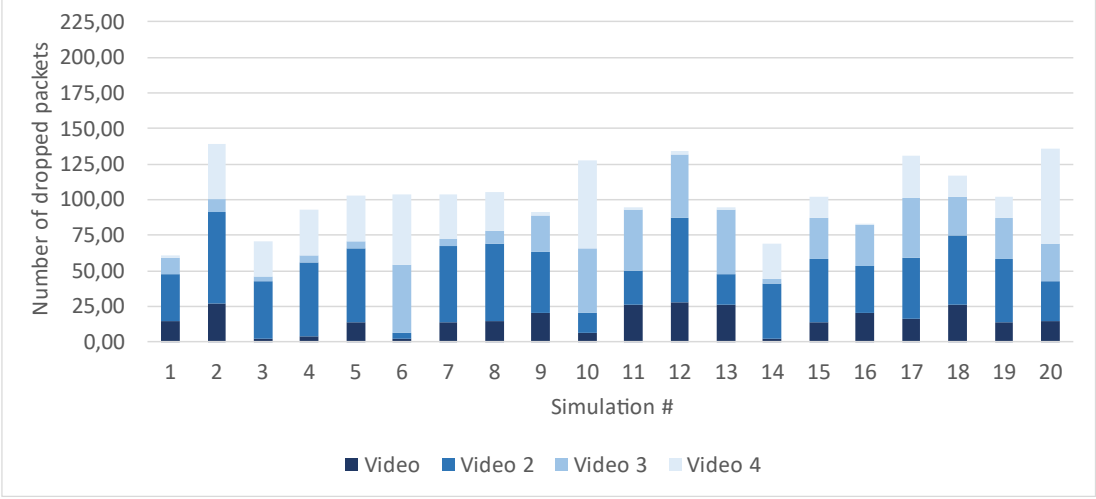

Figure 5-4 - Per flow dropped packets for DRL priorities solution

5.3 Per flow delay performance

By looking at the per flow mean delay across all simulations it is possible to see that the DRL solution is able to achieve much lower delay values which is crucial to the reduction of the number of dropped packets. The intricate relationship between mean delay and dropped packets is discussed in the next segment, but for now, it is possible to also see that the DRL solution was able to provide a much more equitable delay performance across all flows and also mean delay values that are much close to the requirements of each flow.

Although the DRL solution achieved a clear performance boost, there is still not a complete alignment between the mean delay values of each flow and its delay requirements. For example, video 2 achieved an average mean delay bigger than video despite its delay requirements being of a significantly lower delay.

Table 5-3 - Per flow mean delay performance comparison

	Video		Video 2		Video 3		Video 4	
	StdDev	Avg D.	StdDev	Avg D.	StdDev	Avg D.	StdDev	Avg D.
Fixed	0.61	2,23	0.41	1.22	0.06	0.79	2.66	10.20
Random	2.60	3.80	1.37	1.50	3.30	5.59	1.11	1.64
DRL	0.35	1.06	0.74	1.54	0.50	1.10	1.16	1.81

In Figure 5-5, Figure 5-6 and Figure 5-7 it is possible to verify the more equitable distribution among flows given by the DRL solution, as well as the noticeable decrease in average mean delay across all simulations.

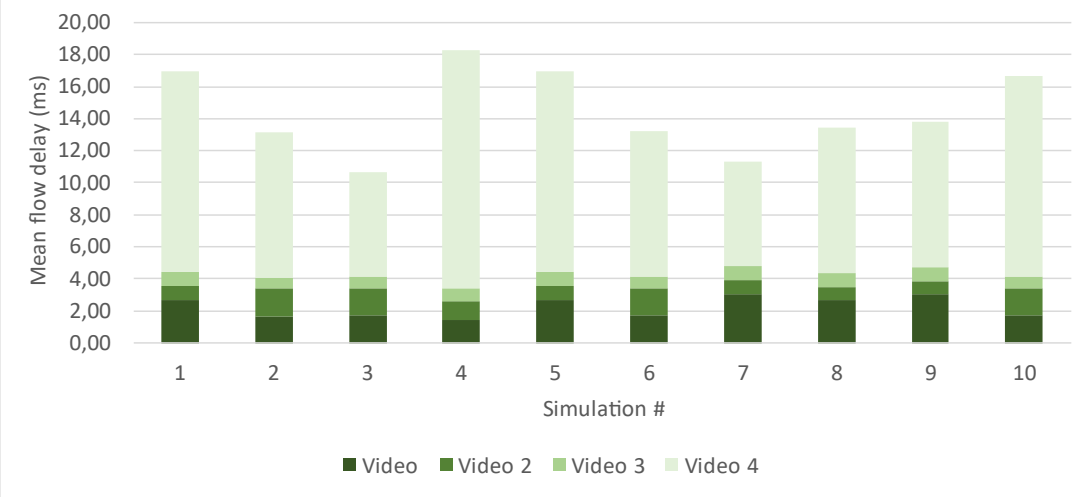

Figure 5-5 - Per flow mean delay for fixed priorities solution

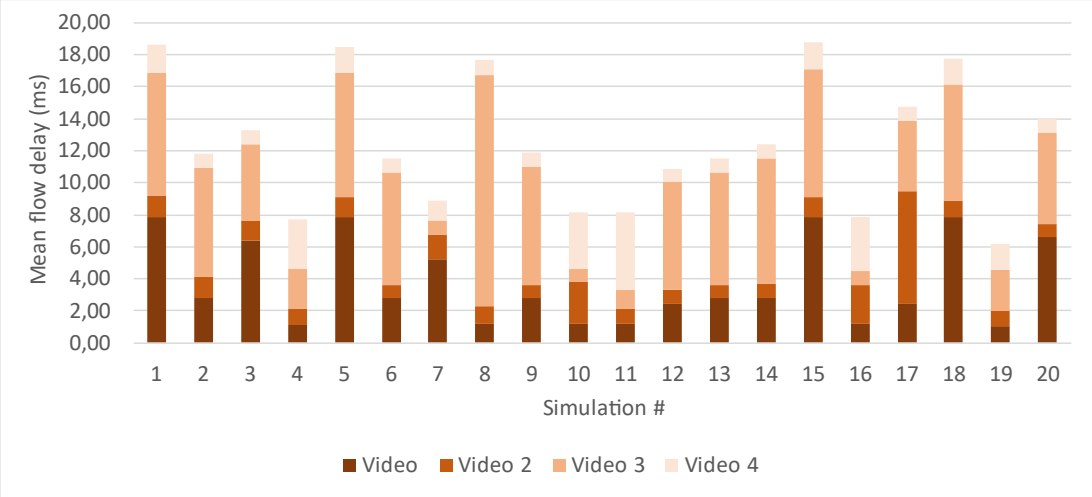

Figure 5-6 - Per flow mean delay for random priorities solution

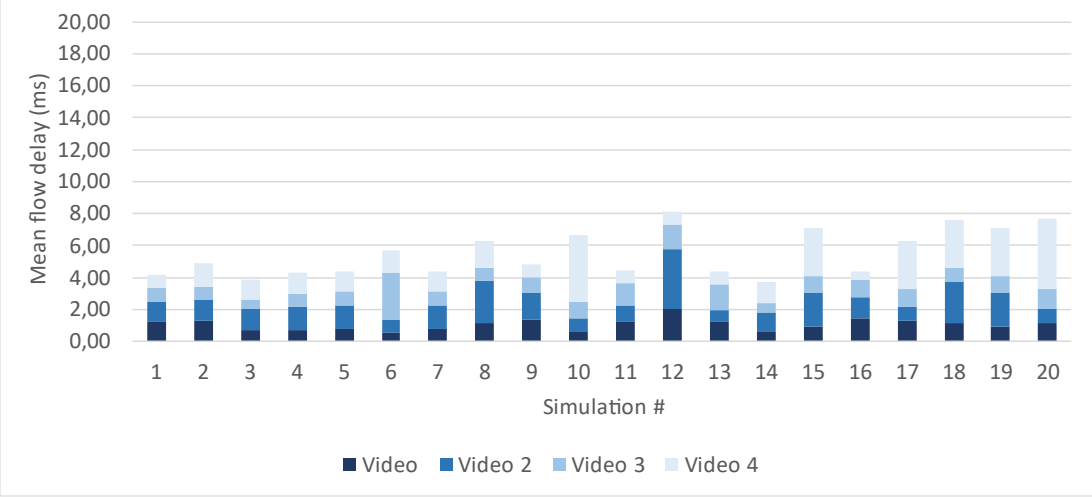

Figure 5-7 - Per flow mean delay for DRL priorities solution

5.4 Delay vs dropped packets ratio

It is important to note that in a deterministic setting the only metric relevant to the application is the number of dropped packets. This number is of course tied to the delay needs of the flow and with a lower mean delay it is probable that the number of dropped packets will also be lower. However, it is possible to maximize this relationship and the goal of a deterministic application should be to minimize the occurrence of dropped packets per unit of mean delay. This can serve as an indicator of how well we are taking advantage of the resources available since there is no need to have a mean delay much lower than the delay requirement if some of the packets are still being dropped. With this in mind a comparison of mean delay values and the number of dropped packets was made.

Table 5-4 - Per flow analysis of mean delay vs dropped packets

	Video		Video 2		Video 3		Video 4	
	Mean delay	Dropped packets	Mean delay	Dropped packets	Mean delay	Dropped packets	Mean delay	Dropped packets
Fixed	2,23	63.4	1.22	58.1	0.79	43.2	10.20	13.5
Random	3.80	44.35	1.50	37.7	5.59	77.15	1.64	20.4
DRL	1.06	15.3	1.54	39.75	1.10	24.25	1.81	23.85

To obtain a metric of performance regarding this balance between mean delay and dropped packets the ratio of dropped packets per millisecond of mean delay was calculated and compared between solutions. This ratio was included in the complete simulation data table that is included in Appendix A.

In Table 5-5 the ratio per flow for each solution is presented. The color code indicates which solution performed better or worse for each flow.

Table 5-5 - Dropped packets per delay unit ratio comparison

	Video Avg ratio	Video 2 Avg ratio	Video 3 Avg ratio	Video 4 Avg ratio
Fixed	29.77	48.31	54.17	1.33
Random	14.40	29.97	16.75	11.06
DRL	13.50	27.42	20.67	13.14

An analysis of the performance differences between each solution aligns with earlier observations in this chapter. The fixed priorities solution significantly enhances the dropped packets per mean delay ratio for the video 4 application, but at the expense of the other three applications. On the other hand, the DRL solution achieves a more balanced distribution of this ratio across flows, though for video 3 and video 4 it falls slightly behind the random priorities' solution. This suggests that while the DRL agent effectively optimizes the mean delay for all four applications, there is room for improvement in minimizing the number of dropped packets.

5.5 Real World vs Simulation

The results and performance obtained during these simulations provide valuable insights into the potential of using DRL for optimizing flow allocation in deterministic networks. However, it is important to acknowledge that there may be differences between the simulation results and the performance that can be achieved in real-life scenarios.

Simulations provide a controlled and reproducible environment where various network conditions and parameters can be manipulated to study their effects on performance. They allow for extensive experimentation and evaluation of different algorithms and techniques without the risks and complexities associated with deploying changes in a live network.

Simulations also have inherent limitations that may affect the generalizability of the results to real-life scenarios. One crucial aspect is the level of fidelity in modeling the network environment. Simulated environments may not fully capture the intricacies, dynamics, and complexities of real-world networks. Factors such as traffic patterns, network topology, and network dynamics can significantly impact the performance of flow allocation algorithms. Therefore, it is important to interpret the simulation results with caution and consider their applicability to real-life scenarios.

Another consideration is the assumption of perfect knowledge and control in simulations. Simulations often assume complete visibility and control over network parameters, which may not be feasible or practical in real-life deployments. It is crucial to account for these uncertainties and design robust algorithms that can adapt and perform well in dynamic and uncertain network conditions.

The scalability of the DRL-based flow allocation framework should be carefully evaluated. Simulations typically involve a limited number of network nodes and flows to facilitate computational efficiency. In real-life scenarios, networks can consist of thousands of nodes and

handle a significantly higher volume of traffic. The performance of the DRL Agent in such large-scale deployments needs to be thoroughly examined to ensure its scalability and efficiency.

To bridge the gap between simulation results and real-life scenarios, future research should focus on validating the performance of the proposed DRL-based flow allocation framework in real-world testbeds or pilot deployments. Conducting experiments in operational networks will provide a more realistic evaluation of the framework's effectiveness, scalability, and robustness. Real-world experiments can also help identify potential challenges and constraints that may arise in practical deployments, leading to further refinements and improvements of the DRL-based priority allocation approach.

5.5.1 Training in the real world

In the context of deploying a DRL Agent in real-world network environments, it is crucial to ensure that the training process does not interfere with the live operational functionality of the network.

To address this concern, it is necessary to design a training framework that allows for the seamless integration of the DRL Agent without disrupting the network's ongoing operations.

One approach is to create a parallel training environment that mirrors the network environment but operates independently. This separate environment could be similar to the one used in this dissertation using OMNET++ and enables the training of the DRL Agent on historical or simulated data, eliminating the risk of adverse effects on the live network.

By training the agent in this isolated environment, it can learn and improve its decision-making abilities without interfering with the actual network's performance.

Once the training is complete, the trained agent can be deployed to the live network, where it can leverage its learned policies to make intelligent decisions and optimize network performance.

6

CONCLUSIONS AND FUTURE WORK

6.1 Future work

The findings in this dissertation are a part of a general effort to learn more about the effectiveness of these types of solutions to networking problems.

Future work is needed to address different types of Algorithms and networks and therefore gain a better understanding of the applicability of these solutions.

6.1.1 Different types of DRL

In order to expand upon the findings of this dissertation, it would be worthwhile to explore the performance of alternative DRL algorithms beyond the standard DQN utilized. While the DQN algorithm has proven to be effective in various domains, there are other advanced DRL techniques that could offer additional insights and potentially improve upon the results obtained. For instance, algorithms like Double DQN, Dueling DQN, or Rainbow DQN introduce modifications to the basic DQN framework that address specific limitations, such as overestimation bias or exploration-exploitation trade-offs.

By systematically comparing the performance of different DRL algorithms, we can gain insights into their respective strengths and weaknesses, as well as uncover new opportunities for optimizing training efficiency, stability, and generalization capabilities in our specific research domain.

6.1.2 Different networks

The exploration of different network topologies can have a significant impact on the study of priority allocation in several ways.

By testing priority allocation algorithms on alternative network topologies, it is possible to evaluate their effectiveness in different network structures. Different topologies introduce varying levels of connectivity, node distribution, and communication patterns. By examining the performance of priority allocation algorithms across diverse topologies, it is possible to assess their robustness, scalability, and ability to handle different network characteristics.

Real-world networks often exhibit heterogeneity, where nodes have different capabilities, resources, or traffic demands. Exploring different network topologies can allow the study of the impact of heterogeneity on priority allocation algorithms. For example, scale-free networks with highly connected hubs may require adaptive algorithms that consider the influence of high-degree nodes on priority assignment. Hierarchical networks with different levels of organization may necessitate algorithms that can handle prioritization within and between levels.

Different network topologies exhibit varying levels of fault tolerance and resilience. By exploring alternative topologies, it is possible to evaluate how priority allocation algorithms handle and recover from node or link failures. For example, small-world networks with their short average path lengths may require algorithms that efficiently handle rerouting and prioritize critical traffic during network disruptions.

Although the dumbbell network topology has been widely used in the evaluation of network protocols and algorithms, it is important to acknowledge its limitations and consider exploring alternative network topologies in future experiments. The dumbbell network, with its simple and symmetric structure, may fail to capture the complexity and heterogeneity found in real-world networks. By experimenting with different network topologies, such as scale-free networks, small-world networks, or hierarchical networks, we can gain a more comprehensive understanding of the behavior and performance of the protocols or algorithms being evaluated.

Scale-free networks exhibit a power-law degree distribution, where a few nodes have a significantly higher number of connections compared to others. This structure accurately represents the presence of highly connected hubs observed in many real-world networks. These networks can be found in various domains, such as social networks, biological networks, and the internet. By studying protocols or algorithms on scale-free networks, it is possible to better understand their robustness to targeted attacks, information spreading dynamics, and the impact of highly influential nodes.

Small-world networks have a high clustering coefficient like regular networks but also exhibit short average path lengths. This combination of local clustering and global connectivity

allows for efficient communication and information diffusion. Small-world networks are commonly used to model social networks, online communities, and peer-to-peer networks. By investigating protocols or algorithms on small-world networks, it is possible to analyze their performance in terms of information dissemination, navigation, and epidemic spreading.

Hierarchical networks exhibit a layered or nested structure, where nodes are organized into groups or levels. This organization provides a natural representation of systems with different levels of organization or authority. These networks can be found in organizational structures, transportation networks, and biological systems. By examining protocols or algorithms on hierarchical networks, it is possible to explore issues related to coordination, decision-making, resource allocation, and the propagation of information across different levels.

Random geometric networks are constructed based on the physical proximity of nodes. Nodes are placed randomly in a space, and connections are established between nearby nodes. This topology captures the concept of spatial relationships in networked systems. Random geometric networks are often used to model wireless ad hoc networks, sensor networks, and mobile networks. Investigating protocols or algorithms on these networks allows us to evaluate issues such as coverage, connectivity, energy efficiency, and spatial routing.

While the alternative network topologies mentioned capture certain aspects of real-life networks, it's important to note that no single topology can fully represent the complexity and diversity found in all real-life networks.

6.1.3 Different traffic patterns

Although this dissertation utilized a fixed traffic pattern and set of flows with fixed configurations for simulating the network, it is important to recognize the potential for future research in simulating more complex traffic configurations and sets of flows.

As network environments continue to evolve, with the proliferation of diverse applications and emerging technologies, there is a growing need to simulate realistic and dynamic traffic scenarios to accurately evaluate network performance.

Future research can explore the incorporation of more complex traffic patterns, such as multimedia traffic and time-varying demands to better mimic the real-world network conditions. Additionally, introducing varied sets of flows with diverse characteristics, including different application types, traffic volumes, and QoS requirements, can provide a more comprehensive understanding of how the network behaves under different traffic scenarios.

By simulating these complex traffic configurations and sets of flows, it will be possible to investigate the impact on packet loss and develop more robust and efficient solutions to the flow allocation problem.

6.2 Conclusions

Deterministic networking has emerged as a crucial topic in the field of networking, driven by the need for highly reliable and predictable network operations. By eliminating the uncertainties associated with traditional networking approaches, deterministic networks offer enhanced performance, reduced packet delays, and improved Quality of Service (QoS) guarantees. With the advent of advanced hardware technologies and the increasing demand for low-latency and high-throughput applications, deterministic networking has become the logical next step for many industries.

In this dissertation, we addressed the Intelligent flow scheduling problem in deterministic networks using DRL techniques. Optimizing flow scheduling in deterministic networks poses significant challenges. The complexity of network traffic patterns, varying flow demands, and the need for efficient resource allocation necessitate novel solutions. This is where DRL presents itself as a promising approach. By training an agent to learn optimal flow allocation policies through interactions with the network environment, DRL can adapt and make intelligent decisions based on current network conditions and requirements. Our research focused on optimizing network performance by leveraging the capabilities of DRL to make intelligent decisions regarding flow priority allocation. A decision was also made to focus on Asynchronous scheduling, an area of research that is especially lacking in research of this kind, although since the start of this dissertation some papers have been released on the topic.

As discussed in the state-of-the-art chapter deterministic networking is a fairly new topic that has seen a surge of research and work in the development of effective protocols and innovative solutions. The use of AI models, more specifically DRL, is still in its early stages as it was shown by the small number of papers done in this field. This landscape still changed quite a bit since the work on this dissertation started in 2022.

The surge of new work makes it very difficult to keep up with new advancements, but an effort was made to integrate in this dissertation most of the work made until the end of development of the dissertation in 2023.

There were major challenges to overcome during the development of this dissertation, as explained, the work on deterministic networking is fairly new and when simulating a

network, we still won't find pre-made modules of all the new scheduling solutions, mainly the UBS used for this dissertation. The setup of the network to behave as needed for the research was then the first challenge to overcome which required a learning curve with OMNET++ and INET modules. The second challenge was to establish the connectivity between the OMNET++ simulation and the python based DRL Agent, this was done with an intricate communication system using JSON files and required the collection of data points from many places in the network. The final challenge was to optimize the learning process of the DRL Agent with little time and data available. Not all possible combinations of parameters were tried, and the volume of data used to train could have been bigger if more time was available. However, with all these challenges it is with a positive note that results were achieved that allow to affirm the applicability of DRL to the scheduling problem in deterministic solutions.

This work fits in the emerging landscape of solutions to tackle the asynchronous scheduling in deterministic networks problem. Most of the recent research papers have shown promising results and this dissertation is aligned with that.

The results obtained from this study have demonstrated the promising potential of DRL as an approach to optimize networking challenges. Even within a limited simulation scenario, the DRL Agent was able to optimize the allocation of resources and achieve a decrease of 42% in the number of packets that arrived above their deterministic delay needs. This is a promising result for the use of DRL, proving that it can work for smaller networks. This research also serves as a foundation for further exploration and advancements, particularly in the utilization of more complex network topologies and advanced DRL models.

BIBLIOGRAPHY

[1] M. Wollschlaeger, T. Sauter e Jasperneite, "The future of industrial communication: Automation networks in the era of the Internet," IEEE Ind. Electron. Mag., 2017.

[2] G. P. Fettweis, "The tactile Internet: Applications and challenges," IEEE Veh. Technol. Mag., 2014.

[3] M. Maier, M. Chowdhury, B. P. Rimal e D. P. Van, "The tactile internet: Vision, recent progress and open challenges," IEEE Commun. Mag., 2016.

[4] S. Samii e H. Zinner, "Level 5 by layer 2: Time-sensitive networking for autonomous vehicles," IEEE Commun. Stand. Mag., 2018.

[5] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang e X. Shao, "Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research," IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 21, NO. 1, FIRST QUARTER, 2019.

[6] N. Finn e P. Thubert, "Deterministic Networking Problem Statement," Internet Engineering Task Force (IETF), 2019.

[7] IEEE Standard 802.1Q, "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks," IEEE , 2014.

[8] "IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications in Bridged and Synchronization for Time-Sensitive Applications in LAN," IEEE Standard 802.1AS, 2011.

[9] J. Prados-Garzon e T. Taleb, "Asynchronous Time-Sensitive Networking for 5G Backhauling," Oulu University and Sejong University, 2021.

[10] IEEE Standard 802.1Qbv, "IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic," IEEE, 2016.

[11] IEEE Standard 802.1Qbu, "IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 26: Frame Preemptio," IEEE, 2016.

[12] IEEE 802.1Qch, "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks–Amendment 29: Cyclic Queuing and Forwarding," IEEE, 2017.

[13] R. M. Badia e J. Cortadella, "High-Level Synthesis of Asynchronous Systems: Scheduling and Process Synchronization," Polytechnic University of Catalonia, Barcelona, 1993.

[14] J. Specht e S. Samii, "Urgency-Based Scheduler for Time-Sensitive Networks," em *28th Euromicro Conference on Real-Time Systems*, 2016.

[15] J. Specht e S. Samii, "Synthesis of Queue and Priority Assignment for Asynchronous Traffic Shaping in Switched Ethernet," em *IEEE Real-Time Systems Symposium*, 2017.

[16] Z. Zhou, Y. Yan, M. Berger e S. Ruepp, "Analysis and Modeling of Asynchronous Traffic Shaping in Time Sensitive Networks," Technical University of Denmark, 2018.

[17] A. Nasrallah e Z. Alharbu, "Performance Comparison of IEEE 802.1 TSN Time Aware Shaper (TAS) and Asynchronous Traffic Shaper (ATS)," IEEE, Yancheng, 2019.

[18] J. Prados-Garzon e Chinchilla-Romero, "Asynchronous Time-Sensitive Networking for Industrial Networks," em *Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, Granada, 2021.

[19] Y. Qiu, J. Ke, C. Liang, Y. Jia, W. Xu e H.-H. Chen, "Deterministic Asynchronous Scheduling with Probabilistic Reliability Guarantee in Industrial Wireless Networks," Cheng Kung University, Cheng Kung, 2021.

[20] J. Prados-Garzon, T. Taleb e M. Bagaa, "Optimization of Flow Allocation in Asynchronous Deterministic 5G Transport Networks by Leveraging Data Analytics," *IEEE TRANSACTIONS ON MOBILE COMPUTING,* vol. 22, nº 3, 2023.

[21] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang e D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE COMMUNICATIONS SURVEYS & TUTORIALS,* vol. 21, nº 4, 2019.

[22] L. Aristotelis, F. Anestis e V. Ioannis, "Deep Reinforcement Learning: A State-of-the-Art Walkthrough," *Journal of Artificial Intelligence Research,* nº 69, pp. 1421-1471, 2020.

[23] X. Wang e H. Yao, "Deep Reinforcement Learning aided No-wait Flow Scheduling in Time-Sensitive Networks," em *IEEE Wireless Communications and Networking Conference (WCNC)*, Beijing, 2022.

[24] H. Yu, T. Taleb e J. Zhang, "Deep Reinforcement Learning based Deterministic Routing and Scheduling for Mixed-Criticality Flows," *JOURNAL OF LATEX CLASS FILES,* 2022.

[25] X. He, X. Zhuge, F. Dang, W. Xu e Z. Yang, "DeepScheduler: Enabling Flow-Aware Scheduling in Time-Sensitive Networking," School of Software and BNRist, Tsinghua University, Tsinghua, 2021.

[26] J. Prados-Garzon, L. Chinchilla-Romero e P. Munoz, "Leveraging DRL for Traffic Prioritization in 5G and Beyond TSN-based Transport Networks," Research Center on Information and Communication Technologies, University of Granada, Granada, 2022.

[27] J. Zhu, Y. Song, D. Jiang e H. Song, "A New Deep-Q-Learning-Based Transmission Scheduling Mechanism for the Cognitive Internet of Things," *IEEE Internet of Things Journal,* 2017.

[28] J. Prados-Garzon, T. Taleb e M. Bagaa, "LEARNET: Reinforcement Learning Based Flow Scheduling for Asynchronous Deterministic Networks," Aalto University, Espoo, Finland; University of Oulu, 90570 Oulu, Finland.

[29] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu e D. Yang, "Experience-driven Networking: A Deep Reinforcement Learning based Approach," em *IEEE Conference on Computer Communications*, 2018.

[30] Z. Xu, K. Wu, W. Zhang, J. Tang, Y. Wang e G. Xue, "PnP-DRL: A Plug-and-Play Deep Reinforcement Learning Approach for Experience-Driven Networking," *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS,* vol. 39, nº 8, 2021.

[31] L. Xu, Q. Xu, J. Tu, Y. Zhang, C. Chen, J. Zhang e X. Guan, "Learning-Based Scalable Scheduling and Routing Co-Design With Stream Similarity Partitioning for Time-Sensitive Networking," *IEEE INTERNET OF THINGS JOURNAL,* vol. 9, nº 15, 2022.

[32] J. Heaton, Introduction to Neural Networks with Java, 2nd Edition, Heaton Research, 2008.

[33] S. Krishnan, "How do determine the number of layers and neurons in the hidden layer," Medium, 8 09 2021. [Online]. Available: https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3. [Acedido em 12 01 2023].

[34] Y. Zhang, J. Wu, M. Liu e A. Tan, "TSN-based routing and scheduling scheme for Industrial Internet of Things in underground mining," *Engineering Applications of Artificial Intelligence,* nº 115, 2022.

[35] G. Stampa, M. Arias, D. Sánchez-Charles, V. Muntés-Mulero e A. Cabellos, "A Deep-Reinforcement Learning Approach for Software-Defined Networking Routing Optimization," Universitat Politècnica de Catalunya, 2017.

[36] M. Bjorklund, "YANG-A data modeling language for the network configuration protocol (NETCONF)," IETF, 2010.

[37] M. Holness, "IEEE Draft Standard for Local and Metropolitan Area Networks-Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment: YANG Data Model," IEEE Standard, 2016.

[38] IEEE Standard 802.1Qat, "IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)," IEEE, 2010.

[39] IEEE Standard 802.1Qcc, "IEEE Draft Standard for Local and Metropolitan Area Networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements," IEEE, 2017.

[40] IEEE Standard 802.1CB, "IEEE Standard for Local and Metropolitan Area Networks–Frame Replication and Elimination for Reliability," IEEE, 2017.

[41] IEEE 802.1Qca, "IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 24: Path Control and Reservation," IEEE, 2016.

[42] IEEE Standard 802.1Qci, "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing," IEEE, 2017.

[43] F. Sagstetter, "Generalized Asynchronous Time-Triggered Scheduling for FlexRay," *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS,* nº 36, 2017.

[44] E. Liang e Z. Wu, "RLlib Flow: Distributed Reinforcement Learning is a Dataflow Problem," em *Conference on Neural Information Processing Systems*, 2021.

[45] W.-X. Liu e J. Lu, "DRL-PLink: Deep Reinforcement Learning With Private Link Approach for Mix-Flow Scheduling in Software-Defined Data-Center Networks," *IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT,* vol. 19, nº 2, 2022.

[46] Y. Tang e H. Guo, "Flow Splitter: A Deep Reinforcement Learning-Based Flow Scheduler for Hybrid Optical-Electrical Data Center Network," Beijing University of Posts and Telecommunications, Beijing, 2019.

[47] A. Nahhas, A. Kharitonov e K. Turowski, "Deep Reinforcement Learning Techniques for Solving Hybrid Flow Shop Scheduling Problems: Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor-Critic (A3C)," em *Hawaii International Conference on System Sciences*, Hawaii, 2022.

[48] Y.-S. Jung, H. Nagasawa e N. Nishiyama, "Extension of Deterministic Scheduling to Stochastic Scheduling," University of Osaka Prefecture. Series A, Engineering and natural sciences, Osaka, 1992.

[49] Y. Peng, A. Jolfaei e K. Yu, "A Novel Real-Time Deterministic Scheduling Mechanism in Industrial Cyber-Physical Systems for Energy Internet," *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS,* vol. 18, nº 8, 2022.

[50] Y. Lu, L. Yang, S. X. Yang e Q. Hua, "An Intelligent Deterministic Scheduling Method for Ultralow Latency Communication in Edge Enabled Industrial Internet of Things," *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS,* vol. 19, nº 2, 2023.

[51] Y. Zhang, Q. Xu, L. Xu e C. Chen, "Efficient Flow Scheduling for Industrial Time-Sensitive Networking: A Divisibility Theory-Based Method," *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS,* vol. 18, nº 2, 2022.

[52] Y. Li, J. Jiang e S. H. Hong, "Joint Traffic Routing and Scheduling Algorithm Eliminating the Nondeterministic Interruption for TSN Networks Used in IIoT," *IEEE INTERNET OF THINGS JOURNAL,* vol. 9, nº 19, 2022.

[53] V. Gavrilut e P. Pop, "Scheduling in Time Sensitive Networks (TSN) for Mixed-Criticality Industrial Applications," Technical University of Denmark, Lyngby, Denmark, 2018.

[54] J. Haxhibeqiri, X. Jiao e P. A. Campos, "To Update or not: Dynamic Traffic Classification for High Priority Traffic in Wireless TSN," Ghent University, Ghent, 2023.

[55] G. R. Ghosal e D. Ghosal, "A Deep Deterministic Policy Gradient Based Network Scheduler For Deadline-Driven Data Transfers," University of California, Berkeley, California, 2020.

[56] W. Han, F. Guo e X. Su, "A Reinforcement Learning Method for a Hybrid Flow-Shop Scheduling Problem," Department of Airborne Vehicle Engineering, Naval Aviation University, Yantai 264001, China, 2019.

[57] X. Wang, H. Yao e T. Mai, "Reinforcement Learning-Based Particle Swarm Optimization for End-to-End Traffic Scheduling in TSN-5G Networks," *IEEE/ACM TRANSACTIONS ON NETWORKING,* 2023.

[58] S. Gracla, E. Beck, C. Bockelmann e A. Dekorsy, "Learning Resource Scheduling with High Priority Users using Deep Deterministic Policy Gradients," Dept. of Communications Engineering, University of Bremen, Bremen, 2022.

[59] F. Grumbach, A. Müller, P. Reusch e S. Trojahn, "Robust-stable scheduling in dynamic flow shops based on deep reinforcement learning," Journal of Intelligent Manufacturing, 2022.

[60] S. Zhang, X. Gong, P. Wang, L. Wang, T. Wang, X. Que e Y. Tian, "Deterministic Transmittable Time-based Asynchronous Scheduler for Fronthaul Networks," em *IEEE Wireless Communications and Networking Conference (WCNC)*, Beijing, 2019.

[61] Z. Zhou e M. S. Berger, "Insight into the IEEE 802.1 Qcr Asynchronous Traffic Shaping in Time Sensitive Network," *Advances in Science, Technology and Engineering Systems Journal,* vol. 4, nº 1, 2019.

[62] D. Yang, Z. Cheng e W. Zhang, "Burst-Aware Time-Triggered Flow Scheduling With Enhanced Multi-CQF in Time-Sensitive Networks," *IEEE/ACM TRANSACTIONS ON NETWORKING,* 2023.

[63] D. Y. Zongrong Cheng, W. Zhang, J. Ren, H. Wang e H. Zhang, "DeepCQF: Making CQF Scheduling More Intelligent and Practicable," School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, 2022.

[64] Y. Zhou, Z. M. Fadlullah, B. Mao e N. Kato, "A Deep-Learning-Based Radio Resource Assignment Technique for 5G Ultra Dense Networks," IEEE, 2018.

[65] Y. Yang, Y. Li, K. Li, S. Zhao e R. Chen, "DECCO: Deep-Learning Enabled Coverage and Capacity Optimization for Massive MIMO Systems," University of Nebraska - Lincoln, 2018.

[66] Y. Dai, D. Xu e K. Zhang, "Deep Reinforcement Learning and Permissioned Blockchain for Content Caching in Vehicular Edge Computing and Networks," IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, 2020.

[67] H. Yu, Taleb e Tarik, "Towards Supporting Holographic Services over Deterministic 6G Integrated Terrestrial & Non-Terrestrial Networks," IEEE, 2023.

[68] P. Schulz, "Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture," IEEE, 2017.

[69] Z. Zhou, Y. Yan, M. Berger e S. Ruepp, "Analysis and Modeling of Asynchronous TrafficShaping in Time Sensitive Networks," Technical University of Denmark, Kongens Lyngby, 2018.

A

Simulation Results

	Video				Video 2				Video 3				Video 4			
	Dropped packets	Mean delay (ms)	Mean delay Std. Dev. (ms)	DP/MD ratio	Dropped packets	Mean delay (ms)	Mean delay Std. Dev. (ms)	DP/MD ratio	Dropped packets	Mean delay (ms)	Mean delay Std. Dev. (ms)	DP/MD ratio	Dropped packets	Mean delay (ms)	Mean delay Std. Dev. (ms)	DP/MD ratio
Best fixed allocation	68	2,675	1,918	25,421	46	0,870	0,339	52,874	52	0,860	0,453	60,465	12	12,560	9,017	0,955
	60	1,684	1,141	35,629	76	1,695	0,754	44,838	36	0,706	0,318	50,992	14	9,078	4,533	1,542
	61	1,696	1,144	35,967	79	1,716	0,752	46,037	35	0,722	0,355	48,476	8	6,518	1,617	1,227
	45	1,427	0,905	31,535	57	1,190	0,809	47,899	39	0,804	0,695	48,507	27	14,870	6,908	1,816
	68	2,671	1,924	25,459	46	0,868	0,338	52,995	52	0,860	0,453	60,465	12	12,560	9,017	0,955
	61	1,696	1,144	35,967	79	1,745	0,757	45,272	35	0,722	0,355	48,476	14	9,078	4,533	1,542
	71	3,064	1,863	23,172	46	0,870	0,339	52,874	48	0,832	0,435	57,692	8	6,518	1,617	1,227
	68	2,675	1,918	25,421	38	0,796	0,343	47,739	52	0,860	0,453	60,465	14	9,078	4,533	1,542
	71	3,064	1,863	23,172	38	0,796	0,343	47,739	48	0,832	0,435	57,692	14	9,078	4,533	1,542
	61	1,696	1,144	35,967	76	1,695	0,754	44,838	35	0,722	0,355	48,476	12	12,560	9,017	0,955
Random priorities	64	7,883	2,700	8,119	45	1,261	1,376	35,686	65	7,754	4,268	8,383	41	1,691	1,628	24,246
	49	2,811	1,584	17,432	37	1,285	0,748	28,794	127	6,841	4,142	18,565	4	0,844	0,332	4,739
	64	6,372	1,809	10,044	16	1,229	0,874	13,019	131	4,815	2,465	27,207	4	0,855	0,330	4,678
	26	1,145	1,327	22,707	46	0,978	0,487	47,035	65	2,525	1,786	25,743	15	3,033	5,441	4,946
	64	7,883	2,700	8,119	41	1,215	1,382	33,745	65	7,754	4,268	8,383	41	1,597	1,426	25,673
	49	2,811	1,584	17,432	22	0,825	0,374	26,667	106	7,034	3,902	15,070	4	0,855	0,330	4,678
	60	5,191	3,197	11,558	50	1,556	1,201	32,134	20	0,859	1,358	23,283	32	1,309	0,538	24,446
	15	1,177	1,401	12,744	51	1,124	0,638	45,374	51	14,427	6,101	3,535	9	0,952	0,490	9,454
	49	2,811	1,584	17,432	18	0,822	0,358	21,898	128	7,399	3,477	17,300	4	0,855	0,330	4,678
	15	1,169	1,406	12,831	54	2,657	1,960	20,324	9	0,837	1,188	10,753	64	3,487	3,233	18,354
	15	1,177	1,401	12,744	37	0,937	0,350	39,488	26	1,197	1,167	21,721	67	4,852	4,164	13,809
	36	2,414	1,764	14,913	22	0,920	0,515	23,913	107	6,707	4,121	15,953	4	0,844	0,332	4,739
	49	2,811	1,584	17,432	18	0,822	0,358	21,898	109	7,025	3,779	15,516	4	0,855	0,330	4,678
	49	2,811	1,584	17,432	24	0,883	0,415	27,180	118	7,816	3,844	15,097	4	0,855	0,330	4,678
	64	7,883	2,700	8,119	50	1,216	1,326	41,118	65	7,949	4,416	8,177	41	1,691	1,628	24,246
	28	1,161	1,319	24,117	61	2,424	2,416	25,165	25	0,886	1,010	28,217	15	3,354	5,694	4,472
	36	2,414	1,764	14,913	60	7,068	5,510	8,489	65	4,395	2,109	14,790	4	0,855	0,330	4,678
	64	7,883	2,700	8,119	36	0,982	0,925	36,660	68	7,292	4,454	9,325	41	1,597	1,426	25,673
	22	1,037	1,305	21,215	48	0,980	0,482	48,980	65	2,535	1,790	25,641	6	1,608	3,886	3,731
	69	6,590	1,927	10,470	18	0,822	0,358	21,898	128	5,743	2,074	22,288	4	0,855	0,330	4,678

	Video				Video 2				Video 3				Video 4			
	Dropped packets	Mean delay (ms)	Mean delay Std. Dev. (ms)	DP/MD ratio	Dropped packets	Mean delay (ms)	Mean delay Std. Dev. (ms)	DP/MD ratio	Dropped packets	Mean delay (ms)	Mean delay Std. Dev. (ms)	DP/MD ratio	Dropped packets	Mean delay (ms)	Mean delay Std. Dev. (ms)	DP/MD ratio
DRL priorities	15	1,225	1,383	12,245	33	1,238	0,797	26,656	11	0,871	1,154	12,629	2	0,816	0,411	2,451
	27	1,294	1,407	20,866	64	1,281	0,756	49,961	9	0,871	1,182	10,333	39	1,442	1,154	27,046
	2	0,680	0,364	2,941	41	1,342	0,788	30,551	3	0,588	0,940	5,102	25	1,271	0,506	19,670
	4	0,691	0,388	5,789	52	1,469	1,165	35,398	5	0,825	1,634	6,061	32	1,300	0,549	24,615
	14	0,808	0,476	17,327	52	1,469	1,165	35,398	5	0,825	1,634	6,061	32	1,257	0,592	25,457
	2	0,564	0,324	3,546	4	0,772	0,249	5,181	48	2,975	3,133	16,134	50	1,382	0,532	36,179
	14	0,803	0,479	17,435	53	1,474	1,162	35,957	5	0,825	1,634	6,061	32	1,257	0,592	25,457
	15	1,170	1,406	12,821	54	2,628	1,965	20,548	9	0,809	1,152	11,125	27	1,650	1,616	16,364
	20	1,347	1,562	14,848	43	1,679	1,254	25,610	26	0,970	1,167	26,804	2	0,850	0,365	2,353
	6	0,609	0,355	9,852	14	0,848	0,399	16,509	46	1,007	1,560	45,680	62	4,206	2,380	14,741
	26	1,199	1,306	21,685	24	1,061	0,720	22,620	43	1,346	1,438	31,947	2	0,816	0,411	2,451
	28	2,023	2,057	13,841	59	3,755	3,909	15,712	45	1,519	1,398	29,625	2	0,816	0,411	2,451
	26	1,192	1,311	21,812	22	0,789	0,369	27,883	45	1,580	1,504	28,481	2	0,816	0,411	2,451
	2	0,660	0,314	3,030	39	1,173	0,788	33,248	3	0,588	0,940	5,102	25	1,292	0,511	19,350
	14	0,921	1,308	15,201	44	2,121	2,432	20,745	29	1,044	1,128	27,778	15	3,033	5,441	4,946
	20	1,452	1,609	13,774	33	1,289	1,074	25,601	29	1,100	2,335	26,364	1	0,535	0,382	1,869
	16	1,288	1,605	12,422	43	0,880	0,352	48,864	42	1,130	0,996	37,168	30	3,000	3,397	10,000
	26	1,145	1,327	22,707	49	2,569	2,490	19,074	27	0,860	0,984	31,395	15	3,033	5,441	4,946
	14	0,921	1,308	15,201	44	2,121	2,432	20,745	29	1,044	1,128	27,778	15	3,033	5,441	4,946
	15	1,178	1,401	12,733	28	0,870	0,318	32,184	26	1,197	1,167	21,721	67	4,435	3,735	15,107