Federal University of São Carlos– UFSCar
Center of Exact Sciences and Technology– CCET
Department of Computer Science– DC
Postgraduate Program in Computer Science– PPGCC

Leandro Cavalcanti de Almeida

Enhancing QoS in Adaptive Video Streaming Through a Smart Closed Loop in Programmable Networks

São Carlos
2024

Leandro Cavalcanti de Almeida

Enhancing QoS in Adaptive Video Streaming Through a Smart Closed Loop in Programmable Networks

Thesis presented to the Postgraduate Program in Computer Science at the Federal University of São Carlos, as part of the requirements for obtaining the title of Ph.D. in Computer Science.

Concentration Area: Distributed Systems and Computer Networks

Supervisor: Fábio Luciano Verdi
Co-supervisor: Rafael Pasquini

São Carlos
2024

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Tese de Doutorado do candidato Leandro Cavalcanti de Almeida, realizada em 08/02/2024.

Comissão Julgadora:

Prof. Dr. Fabio Luciano Verdi (UFSCar)

Prof. Dr. Hermes Senger (UFSCar)

Prof. Dr. Luciano Paschoal Gaspary (UFRGS)

Prof. Dr. Rodrigo Sanches Miani (UFU)

Prof. Dr. Guilherme Piêgas Koslovski (UDESC)

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

This work is dedicated to my loves: Samuel, Lucas, João, Teresa and Paula.

Acknowledgements

To God who extraordinarily guides my life, allowing me to be better every day.

To my dear wife, Paula, whose unwavering support and love are fundamental in my life.

To my sons: Samuel, Lucas, and João, who are the reason behind my life. And with love for my daughter Teresa, who is still within her mother's womb; she is already the most beautiful flower in my garden. Daddy loves you!

In deepest gratitude to my parents, Luciano (*in memoriam*) and Fátima, for their dedication and boundless love throughout my life. Thank you from the bottom of my heart! I love you dearly.

To my sister, Luciana, as well as to all my family members who have provided their unwavering support throughout my journey until this point.

To my supervisor and dear friend, Prof. Fábio Verdi, for the valuable teachings and for making me believe I can go further. My deepest gratitude for your companionship and the attention you gave me. Thank you very much.

To my co-supervisor, Prof. Rafael Pasquini (UFU), for all the insights and valuable comments to improve our research.

To Prof. Chrysa Papagianni (UvA) for her warm welcome into her research group and the invaluable lessons she has imparted to me in such a distinctive manner. Thank you sincerely!

To Prof. Tiago Almeida (UFSCar) for inspiring and motivating me to strive for excellence in my studies. Your guidance and encouragement have been instrumental in my academic journey. Thank you.

To my colleagues at LERIS (UFSCar): André, Gustavo, Guilherme, Washington, Felipe, Renato, Thiago Henrique, Thiago Caproni, Mateus, Alireza and Mina for the camaraderie and the enriching exchange of experiences that have characterized our time together over the past few years. Thank you all for your support and collaboration.

To my colleagues at the University of Amsterdam: Prof. Paola Grosso, Leonardo,

Marco, Floris, Cyril, Angelos, Damian, Lukas, Misha, Jamila, and Peter, for the extraordinary exchange of experiences we've shared. Your contributions have been invaluable, and I'm truly grateful for our collaborative journey.

To IFPB colleagues Prof. Paulo Ditarso, Prof. Thiago Moura, Prof. Jaildo Pequeno, and Prof. Ruan Gomes, to whom I extend my gratitude to all the professors in the informatics academic unit.

To Vladimir Gurevich and Andy Fingerhut (Intel) for their invaluable insights and assistance, which they generously provided whenever required.

To CAPES which financed my sandwich doctorate at the University of Amsterdam.

To the Federal Institute of Paraíba for granting me the opportunity to make a positive impact on the lives of students.

For everyone who directly or indirectly helped me on this journey. Thanks

"Humility is the first step to wisdom."
(Saint Thomas Aquinas)

Abstract

Video traffic constitutes a significant portion of Internet traffic, directly impacting the Quality of Service (QoS) for several applications sharing the network. Emerging on-demand video streaming technologies, like Dynamic Adaptive Streaming over HTTP (DASH), enable a degree of adaptability in video playback to match the quality levels provided by video service providers. However, from the perspective of network providers, monitoring and managing such applications pose considerable challenges due to their client-driven nature. In this work, we address these challenges and present solutions founded on two key pillars: *i*) contemporary programmable networks; and *ii*) artificial intelligence. We propose a solution that encompasses the Monitor-Analyze-Plan-Execute (MAPE) cycle, where monitoring and management mechanisms collaborate to enhance the QoS of DASH video streaming. In this work, we create a "Smart Closed Loop", leveraging the capabilities of the Programmable Data Planes (PDP) and utilizing fine-grained measurements provided by In-band Network Telemetry (INT) to guide Machine Learning (ML) decisions.

We designed and implemented a more precise method for estimating adaptive video service metrics, characterizing significant progress in the field of DASH service monitoring (**M**). Analyzing these estimates (**A**), the Smart Closed Loop can plan (**P**) execution (**E**) strategies within the network infrastructure that aim to deliver the video in better conditions. In this work, the preferred execution strategy is a probabilistic packet discard policy, due to DASH utilizing TCP as a congestion control approach. In this context, we revisited a well-known Active Queue Management (AQM) mechanism based on the RED algorithm, and inspired by it we developed our solution: ingress Random Early Detection (iRED). iRED is a disaggregated P4-AQM fully implemented in programmable data plane hardware (Tofino switches) that saves router resources. This algorithm not only conserves router resources but also aligns with the Low Latency, Low Loss, and Scalable throughput (L4S) framework.

Considering the dynamic nature of video traffic, we design and implement a mechanism

based on Deep Reinforcement Learning to fine-tune iRED parameters in real-time named Dynamic, Enhanced and Smart iRED (DESiRED). With DESiRED, we leverage the benefits attained in enhancing the quality of the DASH video service, making our solution adaptive to the dynamics of network traffic.

Keywords: Programmable Data Plane. In-band Network Telemetry. Machine Learning. Deep Reinforcement Learning. Adaptive Video Streaming.

List of Figures

Figure 1 – The main outcome from the hypothesis of this thesis is the Smart Closed Loop. 29
Figure 2 – Thesis organization. 33
Figure 3 – ABR adaptation logic changes in response to the network conditions. . 36
Figure 4 – PISA abstraction model. Adapted from (Hauser et al., 2023). 37
Figure 5 – In-band Network Telemetry operation. INT metadata is appended on the packets in each hop. In the specific collection point, the monitoring system receives INT metadata. 39
Figure 6 – Types of problem in Supervised Learning. 41
Figure 7 – Neural Network with one hidden layer. 42
Figure 8 – Interaction between the agent and environment in a MDP. Adapted from (Sutton; Barto, 2018). 42
Figure 9 – Deep Q-Network (DQN) high-level workflow (Mnih et al., 2015; Sutton; Barto, 2018). 45
Figure 10 – DQN learning stage workflow (Mnih et al., 2015). 46
Figure 11 – Random Forest feature importance. 57
Figure 12 – Load Patterns. 59
Figure 13 – Real and estimation computed by RF regressor. 59
Figure 14 – The generic architecture of the data plane programmable switch. Headers and Payload follow different paths on the device. 64
Figure 15 – Dual-Queue Active Queue Management (AQM) in Low Latency, Low Loss, and Scalable throughput (L4S) architecture. Adapted from (Schepper; Briscoe; White, 2023). 66
Figure 16 – ingress Random Early Detection (iRED) design. Disaggregating the action of a drop decision reduces wasted resources. 67
Figure 17 – Evaluation setup. Cubic and Prague flows coexist in the same scenario, sharing the programmable switch bandwidth. 71

Figure 18 – iRED resource savings compared to PI2. 72
Figure 19 – iRED resource savings compared to P4-CoDel. 72
Figure 20 – Coexistence evaluation of Cubic and Prague flows (RTT base 10ms). . 73
Figure 21 – Coexistence evaluation of Cubic and Prague flows (RTT base 50ms) . . 74
Figure 22 – DASH experiment. 75
Figure 23 – DASH Results. iRED improves the DASH Quality of Service. 76
Figure 24 – The trade-off: If the Target Delay is small, it can increase packet losses and decrease link utilization. If it is high, increases queueing delays and decreases packet drops. 79
Figure 25 – The Smart Closed Loop overview with DESiRED. At the control plane side, DRL updates the target delay at the data plane. 80
Figure 26 – Operation of the Control Plane in DESiRED involves using fine-grained INT measurements as the input layer for the DQN. Additionally, DASH QoS measurements serve as the basis for calculating agent rewards. . . 81
Figure 27 – Setup of DESiRED Evaluation. 82
Figure 28 – Low Load - Characterized by only ten video player instances managed by WAVE (Load Generator). 83
Figure 29 – High Load. Characterized by forty video player instances managed by WAVE (Load Generator). 84
Figure 30 – Actions performed by the agent in the environment. After the initial random exploration, the agent finds the best target delay value to maximize the QoS of MPEG-DASH. 85
Figure 31 – Model performance results - Decreasing Loss and increasing Reward indicate model convergence. 86
Figure 32 – QoS measurements of the MPEG-DASH video service. DESiRED improves FPS and LBO while minimizing video stall. 88
Figure 33 – QoS estimations of the trained regressor. 89
Figure 34 – QoS estimations of trained regressor and real QoS values. 90
Figure 35 – Topology of experiment. 110
Figure 36 – Add INT metadata. 112
Figure 37 – Wasted Resources (Time). 118
Figure 38 – Stationary Loads. 124
Figure 39 – Non-stationary (Sinusoid) Load. 125
Figure 40 – DESiRED DQN architecture. The input layer is a network of fine-grained measurements, provided by INT. Hidden Layers make up the DQN. The actions are defined in the Output Layer. 125

List of Tables

Table 1 – DASH service metrics estimation. 50
Table 2 – Characteristics used to estimate DASH service metrics. 50
Table 3 – AQM comparative analysis. 51
Table 4 – Main characteristics used to improve DASH QoS/QoE. 53
Table 5 – NMAE for load patterns individually. 58
Table 6 – NMAE for cross evaluations. 60
Table 7 – Configurations 72
Table 8 – Execution percentage at each video quality level. 87
Table 9 – VMs details. 110
Table 10 – Video parameters used in a dashServer. 111
Table 11 – Parameters used for load generators. 112
Table 12 – Load parameters 115
Table 13 – Number of dropped packets 116
Table 14 – Wasted Memory (MB) 117
Table 15 – Latency and Power 119
Table 16 – Wasted Clock Cycles 119
Table 17 – Wasted Weight (Power Consumption) 120
Table 18 – Video parameters used in an MPEG-DASH Server. 123
Table 19 – INT medatada. 123
Table 20 – DQN hyperparameters. 126
Table 21 – DESiRED actions space. 126

List of Acronyms

AQM Active Queue Management

AI Artificial Intelligence

ABR Adaptive Bitrate Streaming

BBR Bottleneck Bandwidth and Round-trip

CC Congestion Control

CNN Convolutional Neural Network

CDN Content Delivery Network

CDF Cumulative Distribution Function

DCTCP Data Center TCP

DASH Dynamic Adaptive Streaming over HTTP

DRL Deep Reinforcement Learning

DQN Deep Q-Network

DNN Deep Neural Network

DL Deep Learning

DESiRED Dynamic, Enhanced and Smart iRED

DT Decision Tree

ECN Explicit Congestion Notification

EWMA Exponentially Weighted Mean Average

FPS Frames per Second

HTTP Hypertext Transfer Protocol

INT In-band Network Telemetry

IETF Internet Engineering Task Force

iRED ingress Random Early Detection

IaC Infrastructure as Code

KNN K-Nearest Neighbors

L4S Low Latency, Low Loss, and Scalable throughput

LBO Local Buffer Occupancy

MAPE Monitor-Analyze-Plan-Execute

ML Machine Learning

MTU Maximum Transmission Unit

MDP Markov Decision Process

MLP Multilayer Perceptron

MSE Mean Squared Error

NN Neural Networks

NMAE Normalized Mean Absolute Error

NPL Network Programming Language

ONT Out-of-band Network Telemetry

P4 Programming Protocol-independent Packet Processors

PISA Protocol-Independent Switch Architecture

PDP Programmable Data Plane

QoS Quality of Service

QoE Quality of Experience

RTT Round-trip time

RL Reinforcement Learning

RF Random Forest

SLA Service Level Agreement

SDN Software Defined Network

SL Supervised Learning

SAND Server and Network-assisted DASH

TCP Transmission Control Protocol

UL Unsupervised Learning

WAVE Workload Assay for Verified Experiments

List of algorithms

1 Decision to drop - Egress 69
2 Action to drop - Ingress 70
3 DESiRED reward policy algorithm. 127

Contents

1 INTRODUCTION 25
1.1 Problem Statement 26
1.2 Hypothesis 27
1.3 Proposal 28
1.4 Research Questions and Contributions 31
1.5 Organization of this Thesis 33

2 FUNDAMENTAL CONCEPTS 35
2.1 Adaptive Bitrate Video Streaming 35
2.2 Network Programmability 36
2.2.1 In-band Network Telemetry 38
2.3 Machine Learning 40
2.3.1 Supervised Learning 40
2.3.2 Reinforcement Learning 42
2.4 Congestion Control 47
2.4.1 Active Queue Management 47
2.5 Summary of the Chapter 48

3 RELATED WORK 49
3.1 Video Service Estimation 49
3.2 AQM in programmable data planes 50
3.3 QoS/QoE improvements in DASH scope 52
3.4 Summary of the Chapter 53

4 USING ML AND INT FOR SERVICE METRICS ESTIMATION 55
4.1 Contextualization and Motivation 55

4.2 Problem Statement . . . 56
4.3 Evaluation . . . 57
4.4 Summary of the Chapter . . . 60

5 MITIGATING NETWORK CONGESTION THROUGH PROBABILISTIC PACKET DROPPING . . . 63
5.1 The Egress Drop Problem . . . 64
5.2 L4S - Low Latency, Low Loss, and Scalable throughput . . . 65
5.3 iRED - ingress Random Early Detection . . . 66
5.4 Evaluation . . . 70
5.4.1 Resource Consumption Analysis - Tofino2 . . . 71
5.4.2 Fair sharing in L4S scope . . . 73
5.4.3 DASH scenario . . . 75
5.5 Summary of the Chapter . . . 76

6 MATERIALIZING A SMART CLOSED LOOP UTILIZING DRL AND INT . . . 77
6.1 The Fixed Target Delay Problem . . . 78
6.2 DESiRED - Dynamic, Enhanced and Smart iRED . . . 79
6.2.1 Data plane operation . . . 80
6.2.2 Control plane operation . . . 80
6.3 Evaluation . . . 82
6.3.1 Case 1 - Using data from the real video player . . . 82
6.3.2 Case 2 - Using the data from the predicted video player . . . 88
6.4 Summary of the Chapter . . . 90

7 FINAL CONSIDERATIONS . . . 91
7.1 Conclusions . . . 91
7.2 Weaknesses and Limitations . . . 92
7.3 Future Directions and Open Issues . . . 93

BIBLIOGRAPHY . . . 95

APPENDIX 103

APPENDIX A – ACHIEVEMENTS . . . 105
A.1 Publications and disseminations during the Ph.D. (2019 - 2024) 105
A.2 Co-authored publications and dissemination during the Ph.D. (2019 - 2024) . . . 106
A.3 Works under review . . . 106

A.4 Other 107

APPENDIX B – EVALUATION DESCRIPTION (SERVICE ESTIMATION) 109

B.0.1 Components description 109

B.0.2 Experiment description 111

APPENDIX C – EVALUATION DESCRIPTION (RESOURCE CONSUMPTION ANALYSIS IN TOFINO2) 115

APPENDIX D – EVALUATION DESCRIPTION (DESIRED) . . 121

D.0.1 Research methodology 121

D.0.2 Environment description 122

D.0.3 Load Pattern 123

D.0.4 Deep Reinforcement Learning mechanism 125

D.0.5 Metrics and Measurements 128

Chapter 1

Introduction

The operation and service management of contemporary networks pose an ongoing horizon of challenges for network administrators. The extensive array of applications and services, each with distinct requirements, inherently introduces a layer of complexity to these environments. Among these applications, on-demand video streaming prominently emerges as a significant component, currently constituting 60-75% of the total Internet traffic (Sandvine, 2023).

In this context, Dynamic Adaptive Streaming over HTTP (DASH)(ISO, 2014) holds a prominent position as the preferred solution adopted by major players in the streaming industry, including Netflix® and Google® (Lederer, 2015). Essentially, DASH serves as a facilitator for video encoding, accommodating a wide range of combinations encompassing resolution, bitrate, Frames per Second (FPS), and various other essential parameters. It provides consumers with a user-friendly, adaptive menu, empowering them to select the most suitable configuration based on their adaptation logic (Bentaleb et al., 2019). This selection process is influenced by the available resources on the consuming devices and the status of the infrastructure. The videos are segmented into uniform-duration chunks, thereby enabling smooth transitions between different video quality levels (Chen; Wu; Zhang, 2015).

Given the importance of adaptive video streaming on the Internet, this subject has captivated the interest of the scientific community (Lin et al., 2020; Wei et al., 2021; Kim; Chung, 2022; Hafez; Hassan; Landolsi, 2023; Spang et al., 2023) with the primary objective to enhance the video Quality of Service (QoS), and consequently the Quality of Experience (QoE). We observed the literature has focused on solutions on the video client side, bringing innovations mainly to the Adaptive Bitrate Streaming (ABR) algorithm. However, we understand that the state of the video player mirrors the conditions of the

network. To a better comprehension, when the network operates without congestion, the ABR algorithm can deliver video content under optimal conditions. During network congestion, the ABR mechanism may offer suboptimal service to users. In summary, we believe that mere adjustments to the ABR algorithm do not directly tackle the fundamental issue. Section 1.1 delves into the research problem, describing the challenges it presents and providing a characterization of the issues that remain unresolved.

1.1 Problem Statement

QoS and QoE in on-demand video streaming play a fundamental role in enhancing viewer satisfaction. As previously mentioned, recent efforts within the community have been dedicated to enhancing ABR algorithms from the client-side perspective. This focus arises from the client-driven nature of DASH service. However, we understand that this approach has some drawbacks.

The fundamental consideration is that leaving the adaptation or adjustment decisions exclusively to ABR serves as a palliative measure to address the issue rooted in network congestion. This situation arises due to the inherent approach of ABR algorithms, which delegate the responsibility of fair distributing network resources among competing users to the transport layer algorithms (Spang et al., 2023).

Moreover, from the point of view of network service providers, it's important to note that: firstly, they lack control over video clients as decisions regarding the download of the next video segments are determined by the ABR; secondly, the greedy policy of ABR decisions can lead to a global deterioration of QoE due to a lack of coordination among clients (Kim; Chung, 2022); and thirdly, ABR solutions tend to be mutually exclusive, making the integration of their respective characteristics into a single video player a formidable challenge. Consequently, network service providers face difficulties in providing a premium QoS to DASH clients.

In this regard, efforts initiated by the IETF in 2013[1] culminated in the proposal of a complex architecture referred to as Server and Network-assisted DASH (SAND) in 2017 (Thomas et al., 2017). Within the SAND architecture, elements are categorized into three groups: *i*) DASH clients, *ii*) DASH-assisting network elements (DANE), and *iii*) conventional network elements. In this scenario, elements *i* and *ii* engage in the exchange of DASH messages to facilitate the efficient delivery of video segments. However, this architecture was too intricate, primarily because it mandated the presence of network elements with the capacity to comprehend DASH messages. This requirement not only introduced additional overhead in network traffic but also led to its limited adoption by service providers. Furthermore, there were inherent challenges linked to the scalability of the architecture, as the management complexity increased proportionally with the growing

[1] https://mpeg.chiariglione.org/about/events/workshop-session-management-and-control-mpeg-dash

number of DASH clients within the environment. Therefore, proposing a solution to help existing Congestion Control (CC) mechanisms enhance the QoS of real-time video services represents a current challenge.

As a presented scenario, the networking community has sought more innovative solutions in the QoS/QoE guarantees of on-demand video streaming. The recent advances in the domain of Programmable Data Plane (PDP)(Bosshart et al., 2014a) and In-band Network Telemetry (INT) (P4, 2021) have conducted a paradigm shift, bringing us the capability to attain granular visibility, discernible on a per-packet basis, effectively altering the scenario of the challenges associated with data availability in the context of monitoring within computer networks. Furthermore, recent progress in the field of Artificial Intelligence (AI) applied to computer networks (Boutaba et al., 2018) has reshaped the mechanisms to support the management of network services by incorporating some level of intelligence. **This thesis is situated within this context, offering contributions to the current state-of-the-art in enhancing the QoS for on-demand video streaming services, leveraging from the killing technologies for monitoring and service operations in programmable networks.** In the next section, we will examine the hypothesis outlined in this work, providing the supporting arguments for our foundations.

1.2 Hypothesis

Conventional practices for monitoring and managing computer networks are predominantly based on the evaluation of a variety of performance metrics. These metrics encompass various parameters, including the utilization of CPU resources by specific equipment, the volume of packets transmitted via network interfaces, and the latency experienced in particular connections. These metrics serve as indicators of the network's condition and are subject to influence by other external factors. For instance, it is anticipated that a direct correlation exists between the number of connections on a web server and the corresponding memory utilization.

In the context of networking, this cause-and-effect association is a well-established concept, frequently utilized by transport protocols for congestion inference. Notably, Transmission Control Protocol (TCP), for example, relies on not receiving acknowledgments to deduce that the network is congested. Meanwhile, the Bottleneck Bandwidth and Round-trip (BBR) protocol employs Round-trip time (RTT) measurements to detect congestions (Cardwell et al., 2016). In this case, the information acquired reflects a historical state of network conditions, signifying that the inferred congestion pertained to a moment before its detection. Consequently, the strategies employed are inherently reactive rather than proactive.

Within switches, the status of buffers holds relevance as they serve as critical indicators

of network congestion (Kim et al., 2018). Furthermore, these buffers' occupancy levels can influence various metrics employed by monitoring systems, including RTT, delay, and jitter. Nonetheless, the real-time acquisition of data from a buffer is a challenge, primarily due to the resource-intensive nature of this task. This challenge arises from the requisite of the use of additional bits in headers and the resultant elevated power consumption in network devices (Arslan; McKeown, 2019).

In the context of modern networks, a notable shift has occurred, making it feasible to collect real-time data from buffers with only minimal increases in power consumption or capacity loss (Arslan; McKeown, 2019). Furthermore, integrating PDP and INT has enabled the more precise and comprehensive observation of network data. We believe that services operating across a network have the potential to use performance metrics as a means to evaluate the QoS. In such scenarios, quality metrics associated with a service may mirror the state of the network on which the service is deployed, as exemplified in (Almeida; Pasquini; Verdi, 2021).

Based on these initial considerations, our core hypothesis is that **fine-grained measurements provided by In-band Network Telemetry (INT) within Programmable Data Planes (PDP) can be used as input layer for Machine Learning (ML) models to drive Active Queue Management (AQM) mechanisms with the ultimate goal to enhance the Quality of Service (QoS) of DASH video services**. In the next section, as we aim to evaluate the hypothesis, we will present our proposal by detailing each component of the solution within the framework of the Monitor-Analyze-Plan-Execute (MAPE) cycle.

1.3 Proposal

To evaluate the feasibility of the hypothesis, we designed a proposal based on the MAPE cycle, where each component plays a defined role in the Smart Closed Loop framework, as can be seen in Fig. 1. In short, fine-grained INT measurements are collected directly from the data plane in-line rate, characterizing the **Monitor** phase. These measurements serve as the input layer for two intelligent components at the control plane: *i*) a regressor utilizing Supervised Learning to infer the state of DASH clients through QoS estimations, thus characterizing the **Analyse** phase; and *ii*) a Deep Reinforcement Learning (DRL) agent that **Plans** the actions and interacts (**Execute**) with the environment (network) to maximize rewards (best QoS). It achieves this by fine-tuning AQM parameters, to improve the quality of the DASH video service. Before explaining how this agent **Plans** the actions, for a better understanding, we will proceed one step further and describe the **Execute** phase. After this, we will step back and show how the **Plan** phase is designed.

The component chosen to implement the **Execute** phase in this work was the prob-

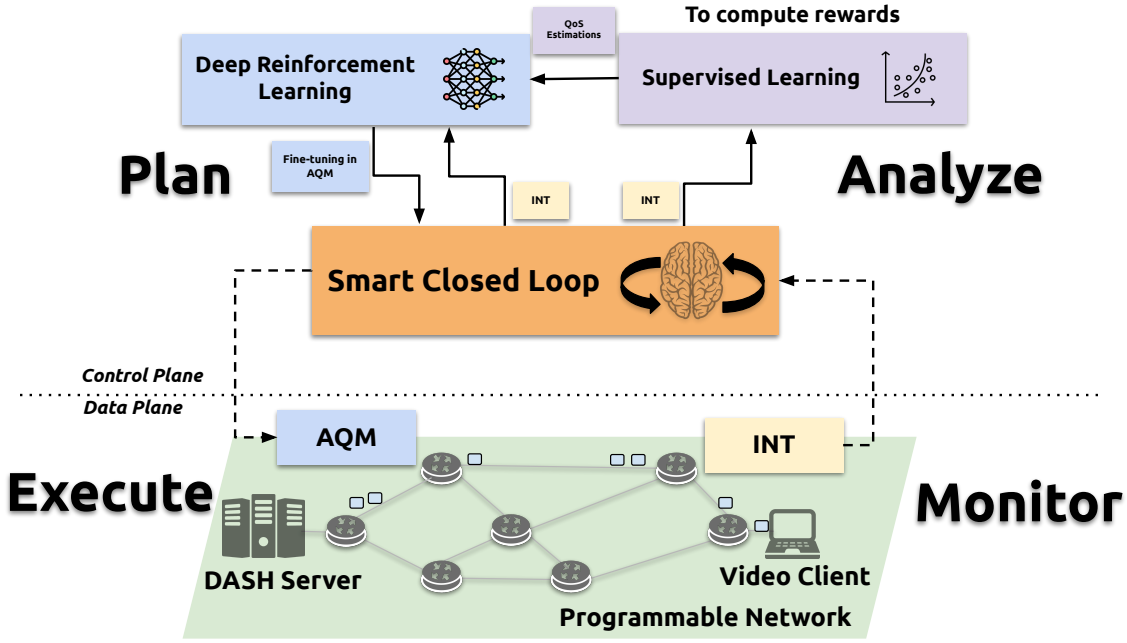


Figure 1 – The main outcome from the hypothesis of this thesis is the Smart Closed Loop.

abilistic packet discard/marking policy. In this context, we revisited a well-known AQM mechanism based on the Random Early Detection (RED) algorithm (Floyd; Jacobson, 1993). The RED mechanism is quite simple: drop packets before the queue buffer is full then avoiding packet tail drop. The idea behind this is that for latency-sensitive applications, such as DASH, the lower the time packets stay in the queue, the better. We argue that, with this strategy, there is a high probability of discarding packets from culprit flows, that is, those that exacerbate congestion. Given that DASH uses TCP as transport layer protocol (ISO, 2014), the CC can benefit from drops, reacting as soon as possible to the congestion signal by receiving duplicate ACKs and triggering the TCP Fast Recovery. This early feedback accelerates the reduction of the packet-sending rate across the network and consequently alleviates the congestion.

The primary methods for conveying congestion conditions to senders include packet marking using Explicit Congestion Notification (ECN) bits and selective packet dropping, which are typically implemented in the Active Queue Management (AQM) solutions (Gombos et al., 2022). Considering the dynamic adaptation logic of DASH (Spang et al., 2023), we believe the video QoS would be minimally affected by discards, as the player maintains a local buffer to temporarily store the last received frames.

We designed and implemented a RED mechanism, called ingress Random Early Detection (iRED), which is a disaggregated P4-AQM fully implemented in a programmable data plane hardware (Tofino switch). To understand the meaning of disaggregation in this work, it's essential to observe the entire path of a packet within a PDP. Initially, incoming packets are received at the Ingress Block and the match-action logic (e.g., IPv4 forwarding) is executed. In the Traffic Manager, which is non-programmable, packets

can be accommodated temporarily in the appropriate output queue. After this, packets are sent to the Egress Block, where AQM algorithms are traditionally implemented as a match-action logic to make drop decisions (set a packet to drop). Finally, the packets marked to be dropped will be effectively discarded in Egress Deparser, which is the last phase in the pipeline.

One may ask why the AQMs are deployed in the Egress block causing a waste of resources since the packet traversed all the pipelines in the switch and then was discarded. It makes much more sense to deploy the AQMs in the Ingress block. However, queuing delay metadata (or queue depth) which is the main information used as input to the AQM algorithm to decide whether the packet should be dropped or not, is captured by the Traffic Manager and made available only in the Egress block. Then, AQM algorithms must be deployed at the Egress block. The challenge is to design a solution in which the packets are dropped in the Ingress block saving resources of the network device. We discuss this topic as **the Egress drop problem**.

In this sense, disaggregation in this work means that iRED takes the decision to discard a packet in the Egress block (after Traffic Manager), but the drop action is performed in the Ingress block. Based on this concept, our evaluations have shown that iRED can save router resources (See Chapter 5).

In addition, our iRED solution supports the recent L4S framework (Briscoe et al., 2023), which is a new architecture proposed by Internet Engineering Task Force (IETF) that enables Internet applications to achieve low queuing latency, low congestion loss, and scalable throughput control (L4S). The L4S architecture introduces incremental changes to both hosts and network nodes. On the host side, L4S incorporates a novel variant of a "Scalable" CC algorithm known as TCP Prague (Briscoe et al., 2018). TCP Prague adjusts its window reduction in proportion to the extent of recently observed congestion. This stands in contrast to "Classic" CC algorithms, which typically implement a worst-case reduction, typically by half, upon detecting any signal of congestion. At network nodes, L4S brings a dual queue coupled mechanism (Schepper; Briscoe; White, 2023), in which one queue is for Classic traffic and another queue is for Scalable traffic. This coupled mechanism allows fair use of bandwidth, ensuring harmonious coexistence between CC flavors. iRED supports the L4S by categorizing traffic as either Classic (subject to dropping) or Scalable (marked with the ECN bit), thus ensuring fairness among various flows through a combined packet dropping and marking in a coupled mechanism.

A key aspect of iRED involves setting an appropriate threshold value, often based on queue delay (target delay) or queue depth. A too-small target delay increases packet losses, raising drop probability and reducing overall link utilization. Conversely, a high target delay results in long queuing delays but a lower drop probability. We discuss this topic as **the fixed target delay problem**. As the DASH traffic is dynamic, it would be

interesting for the target delay to have a dynamic characteristic, adjusting the load applied to the network buffers. In the Smart Closed Loop (Fig. 1), the **Plan** phase is characterized by determining the ideal target delay for the current network load conditions.

In this work, to adjust the iRED target delay dynamically, we design and implement the Dynamic, Enhanced and Smart iRED (DESiRED), a joint solution that leverages the In-band Network Telemetry (INT) and Deep Reinforcement Learning (DRL). With DESiRED, fine-grained measurements are used as an input layer for the Deep Queue Network (DQN) to define management actions, that is, find the ideal target delay based on the current network conditions (INT measurements). Based on the experience of the actions taken, DESiRED learns the ideal target delay values following a rewards policy that maximizes DASH QoS. With DESiRED dynamically adjusting the target delay, we achieved the best results compared to a fixed target delay. We understand this solution materializes the Smart Closed Loop.

Starting from our solution proposal, we enumerate three research questions that guided this work. In the following section, we will introduce and discuss these questions, explaining how we formulated our answers. These answers form the foundation for the five contributions of this work.

1.4 Research Questions and Contributions

In this section, we enumerate five significant contributions that are the answers to the three research questions guiding this thesis.

Question 1: Is there a correlation between the conditions of the network infrastructure and the state of a DASH service? If such a correlation exists, what methodology is best employed to map and quantify this relationship effectively?

To address this first question, we initiate our investigation by exploring the estimation of the quality of service through network state information. This topic has been commonly discussed in the literature (Stadler; Pasquini; Fodor, 2017; Calasans, 2020), enabling a network operator to obtain a clear view of the state of the video application based on the conditions of the network infrastructure.

In our **first contribution**, we characterize this issue as a regression problem that can be solved through supervised learning. In this particular context, our **second contribution** trains a regressor employing fine-grained INT measurements as the input layer for the estimation of DASH metrics. These infrastructure metrics were collected at line rate from the router buffers, offering a detailed view of network congestion status. The outcome derived from this trained regressor yielded improved accuracy, exhibiting a reduced prediction error in comparison to the state-of-the-art solutions. Furthermore, this regressor could be valuable in cases where the network operator lacks access to the application's

performance metrics, enabling awareness of the application's status with a certain level of accuracy.

Question 2: Based on these correlations, is it possible to manage and orchestrate some mechanism in real-time to improve the QoS of a DASH service?

In our initial investigations, we established a demonstrable correlation between the buffer occupancy within routers and the DASH QoS. Drawing upon these findings, our exploration delved deeper into the investigation of techniques designed to mitigate congestion. Potential strategies were analyzed encompassing to ReRouting (Chiesa et al., 2019; Verdi; Luz, 2023; Marques; Levchenko; Gaspary, 2023), Queue Priorization (Rahouti et al., 2021), and AQM (Kundel et al., 2018; Papagianni; Schepper, 2019). In our initial analyses, we opted to employ packet discard as the primary strategy, aimed at efficiently draining packets directly from the queues and consequently alleviating network congestion.

Regarding our **third contribution**, we design and deploy iRED, our implementation of the RED (Floyd; Jacobson, 1993) algorithm, in both, hardware (Tofino) and software (Bmv2). Our evaluations indicated that iRED enhances the QoS for DASH service (Almeida et al., 2022), dropping packets as soon as possible, and signaling the sender to reduce the packet sending rate. Also, to the best of our knowledge, iRED represents the pioneering P4-AQM with support for L4S framework, fully implemented in the data plane. Moreover, it employs a key concept regarding disaggregation, wherein the decision to discard a packet is made in the Egress block (after Traffic Manager). Still, the action is executed in the Ingress block, thus optimizing the utilization of router resources. In this context, we have identified and elucidated a recurring issue in the state-of-the-art AQM mechanisms integrated into the data plane. In this thesis, our **fourth contribution** pertains to the definition and characterization of the "Egress drop problem".

Question 3: Given the flexibility of the programmable data plane and the robustness of reinforcement learning models, is it possible to create a joint solution, encompassing the data plane and the control plane of a network to materialize the MAPE cycle?

In this thesis, we design and implement a solution named DESiRED, that materializes a "Smart Closed Loop" as our **fifth contribution**. With DESiRED, we successfully addressed the "fixed target delay problem" in the context of AQM in a programmable data plane. The target delay is a threshold value commonly used by AQMs to trigger the drop probability for every packet. Historically, this value has remained static, and as a result, we refer to it as the "fixed target delay". We posit that a dynamic target delay could offer advantages for improving the QoS in the context of DASH, given the inherently dynamic nature of video traffic (Spang et al., 2023). This was achieved by dynamically adapting the target delay in real-time, through the integration of In-band Network Telemetry (INT), Service Estimation, DRL, and iRED.

1.5 Organization of this Thesis

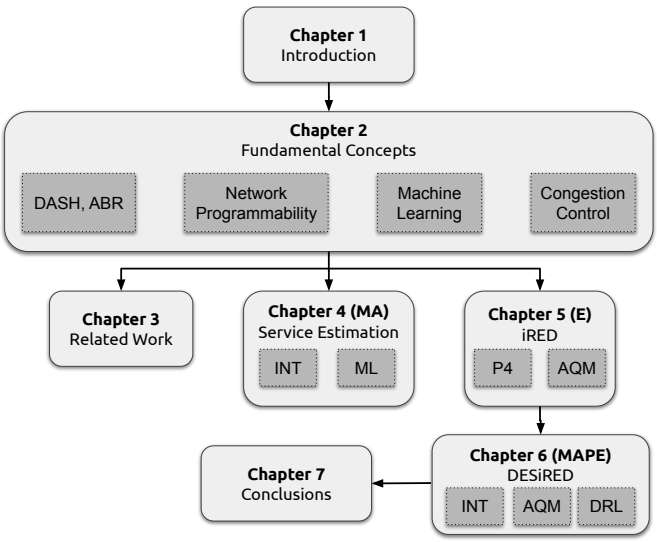


Figure 2 – Thesis organization.

The organization of the chapters, their correlation and the placement of each component within the MAPE cycle are detailed in Fig. 2. After the introduction, the fundamental concepts necessary for an understanding of the topics discussed in this thesis are presented in Chapter 2. In Chapter 3, the related works are briefly described, providing an exposition of the primary distinctions achieved through the advancements introduced within this thesis. Chapter 4 provides a comprehensive account of the challenges faced and the progress achieved in addressing the issue of service metrics estimation, using fine-grained INT measurements as input layer to ML models. Chapter 5 introduces iRED, a P4-based AQM fully implemented within the data plane, designed to support the L4S framework. We illustrate the materialization of the Smart Closed Loop (DESiRED), which encompasses both the control and data planes in Chapter 6. Chapter 7 presents the final considerations, including lessons learned and future directions to be explored.

Chapter 2

Fundamental Concepts

The objective of this chapter is to provide clarity on the conceptual foundations, highlighting how the adaptive video service can be improved from the Smart Closed Loop introduced. In this scenario, our solution can be interpreted as a fusion of network programmability, machine learning and active queue management.

2.1 Adaptive Bitrate Video Streaming

Currently, on-demand streaming video traffic from services such as Netflix and YouTube constitutes a substantial portion, accounting for 60-75% of Internet traffic (Sandvine, 2023). This scenario represents a predominant profile wherein a single application commands the largest share of Internet traffic. In this context, contemporary video applications exert a notable influence on Internet traffic patterns (Spang et al., 2023), potentially impacting the QoS for other applications that share the same network resources.

The delivery of audiovisual content through Hypertext Transfer Protocol (HTTP) has evolved into a standard practice adopted by major players in the video industry (Lederer, 2015). Within this scenario, the integration of the DASH standard alongside ABR algorithms has significantly reshaped the content delivery process.

On the server side, as an integral aspect of the encoding process, the video is partitioned into segments of uniform size, commonly denoted as chunks. Subsequently, each of these segments undergoes encoding at multiple discrete quality levels. Conversely, on the client side, the ABR adaptation logic governs the consumption pattern of video segments (Spiteri; Sitaraman; Sparacio, 2018). As depicted in Fig. 3, the ABR adaptation logic is capable of dynamically transitioning between various video quality levels in response to

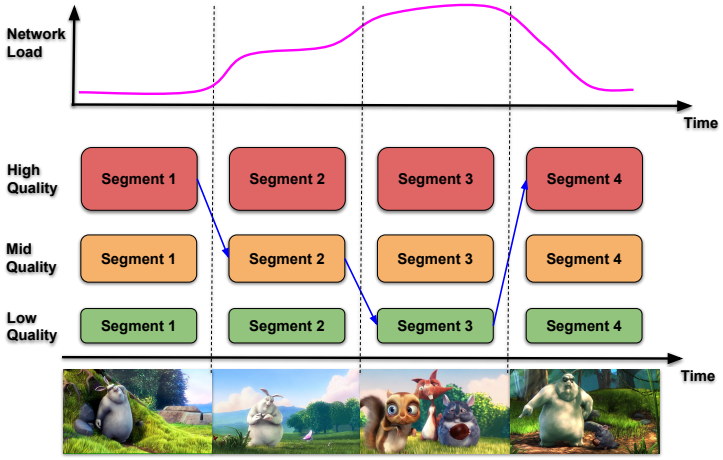


Figure 3 – ABR adaptation logic changes in response to the network conditions.

the network load conditions, with the ultimate goal of optimizing the QoE for a single video session (Spang et al., 2023).

Although this technology is considered a "smart" mechanism due to its capability to adapt to fluctuations caused by network load, unfortunately, such a solution by itself is not enough to keep the pace required by certain Service Level Agreement (SLA) and other auxiliary mechanisms should be designed to jointly work with them. These additional mechanisms can emerge from various avenues, with one notable avenue being network innovations. Network programmability has opened doors to increased flexibility in packet processing within the network infrastructure. This newfound flexibility signifies an essential paradigm shift that can serve as a supplementary mechanism in the provision of video services over the Internet.

2.2 Network Programmability

With the emergence of Software Defined Network (SDN) (Casado et al., 2007; McKeown et al., 2008), the initial advances towards network programmability were taken (Hauser et al., 2023). The separation of data and control planes has brought about increased flexibility and new avenues of research in the field of computer networks.

In this context, the OpenFlow protocol pioneered the provision of a network abstraction layer to the control element (Controller), thereby enabling the configuration and manipulation of the network's data plane through software programming. However, this model still lacked sufficient flexibility for adding new headers and defining new actions after flow matching due to limitations imposed by the rigidity in the intelligence embedded in the pipeline of ASIC processors (Garcia et al., 2018).

Traditionally, the data plane has taken on the role of processing packets according to the logic prescribed by the control plane. However, the game changes with the introduction of data plane programmability, which facilitates the incorporation of intelligence

during packet processing at the hardware's most proximate level, without the necessity for control plane intervention. Noteworthy examples of languages tailored for programming the data plane in this context include Programming Protocol-independent Packet Processors (P4) (Bosshart et al., 2014b) and Network Programming Language (NPL) (Broadcom, 2019). Among these, P4 stands out as a language that has gained wide acceptance within both the scientific and industrial communities.

Beyond the evident flexibility, the response time of the data plane resides in the order of nanoseconds, while the control plane operates on a scale of seconds or milliseconds (Hauser et al., 2023). P4 code exhibits versatility across a range of architectures, each characterized by unique match-action pipeline stages and packet processing attributes. These architectural designs are fundamentally underpinned by an abstraction model, such as Protocol-Independent Switch Architecture (PISA), that facilitates the mapping of instructions and hardware-specific features tailored to individual targets. The P4 language abstracts packet parsing and processing, by providing a generalized forwarding model.

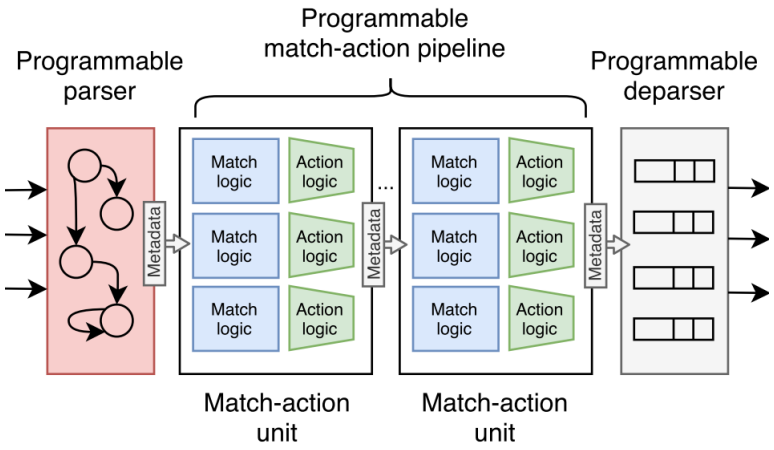


Figure 4 – PISA abstraction model. Adapted from (Hauser et al., 2023).

As illustrated in Fig. 4, the PISA architecture consists of three fundamental components: a programmable parser, a programmable match-action pipeline, and a programmable deparser.

The programmable parser operates as a finite state machine that dictates the order of header extraction from incoming network packets. Once the headers and their associated fields have been exposed by the programmable parser, they can traverse through multiple stages within the match-action pipeline. In this context, tables are defined to effectively manage metadata, adhering to the logic established by the programmer. For instance, in the context of IPv4 routing, a table can be configured to perform matches against the destination address and execute actions such as: i) decrementing the TTL field; ii) adjusting physical addresses; iii) configuring the output port.

The deparser represents the programmable component responsible for specifying the packet's serialization, i.e., reassembly, for subsequent transmission.

In the context of programmable networking, the industry has recently adopted support for the P4 language. Notably, Xilinx has introduced the NetFPGA SUME, an advanced network card that enables P4 programming. Furthermore, industry leaders such as Intel and Broadcom have launched dedicated processors, namely the Tofino (Intel, 2021) and Trident4 (Broadcom, 2023), specifically tailored for the programmable networking market.

For educational purposes, a software-based switch version capable of executing P4 code is available. The Bmv2 (Behavioral Model Version 2) [1] is an integral component within the P4 ecosystem, designed as a tool for the study, testing, and analysis of solutions directly within the data plane.

In the P4 ecosystem, commonly referred to as a "target", each equipment category possesses a specific architecture. In the case of NetFPGA, the employed architecture is the Simple Sume Architecture, whereas Tofino utilizes the Tofino Native Architecture and the Bmv2 is rooted in the V1model architecture (Hauser et al., 2023).

A comprehensive understanding of the architecture supported by the target is essential for the programmer, as it delineates the implementation particulars of the various stages. Furthermore, the availability of metadata for utilization depends on the specific architecture. These metadata can be accessed and integrated into network monitoring systems. For instance, in the V1model architecture, metadata about switch interface buffers can be retrieved and encapsulated within packets during the forwarding process. This technique is elaborated in the INT specification (P4, 2021), designed to facilitate metadata collection within the data plane without necessitating control plane intervention or involvement.

2.2.1 In-band Network Telemetry

Advancements in programmable data plane (Bosshart et al., 2014b) have enabled network devices to autonomously report the network's state, eliminating the need for direct control plane intervention (Arslan; McKeown, 2019). In this scenario, packets incorporate telemetry instructions within their header fields, facilitating the fine-grained collection and recording of network data. The telemetry instructions are defined in the INT data plane specification (P4, 2021).

Figure 5 illustrates the operation of INT within an arbitrary network. The network comprises four hosts, namely *H1, H2, H3*, and *H4*, along with four nodes equipped with P4 and INT support, denoted as *S1, S2, S3*, and *S4*. Each network node possesses a set of metadata, represented by orange (S1), magenta (S2), green (S3), and blue (S4) rectangles. This metadata contains information specific to each node, such as Node ID, Ingress Port, Egress Spec, Egress Port, Ingress Global Timestamp, Egress Global Timestamp, Enqueue Timestamp, Enqueue Queue Depth, Dequeue Timedelta, and Dequeue Queue Depth, as specified in the V1Model architecture.

[1] https://github.com/p4lang/behavioral-model

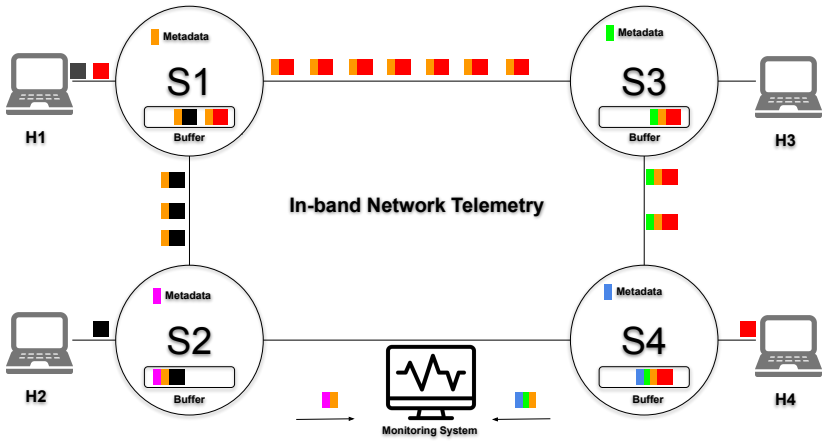


Figure 5 – In-band Network Telemetry operation. INT metadata is appended on the packets in each hop. In the specific collection point, the monitoring system receives INT metadata.

Still in Figure 5, there are two distinct flows depicted: one represented by red packets and the other by black packets. The red flow is required to adhere to the prescribed network path *f1=H1, S1, S3, S4, H4*, while the black flow must traverse the designated path *f2=H1, S1, S2, H2*.

At each network hop along these paths, the data plane of the network devices employs telemetry instructions to facilitate the collection and inclusion of metadata within the packets as they traverse each node. This process is iteratively performed throughout the path, starting from the first node after the source and concluding at the last node before reaching the destination. Upon reaching the destination node, the metadata is extracted from the packet and subsequently relayed to the monitoring system. The original packet is then directed to its final destination.

In addition to the modes delineated in the INT specification, alternative approaches exist for collecting metadata within programmable networks. One such approach involves the utilization of an "exclusive telemetry flow" to monitor the network's state, which, in this work, is referred to as Out-of-band Network Telemetry (ONT).

In the ONT scenario, dedicated probe packets are employed to gather metadata, eliminating the need for any modifications to the data packets associated with the services operating within the network. The primary advantage of this approach lies in its ability to maintain the integrity of application traffic, as it traverses the programmable network without modifications, thereby mitigating issues related to packet growth, such as fragmentation.

Conversely, the use of an exclusive telemetry flow could introduces additional overhead to the overall network traffic. This is due to the necessity of having a dedicated monitoring flow ONT for each service running within the network.

One of the primary advantages of employing telemetry lies in the exceptional level

of granularity it offers. Every individual packet traversing the network carries pertinent information directly to the monitoring system at the line rate. This level of granularity aligns with the perspective presented in (Castro et al., 2019), wherein it is recognized that a substantial volume of data can prove useful for ML algorithms.

2.3 Machine Learning

ML is a field within artificial intelligence where computers are designed to learn from past experiences. In this context, learning occurs through the ability to enhance performance in executing tasks based on past experience (Faceli et al., 2011b). In essence, the aim of ML is to identify and leverage hidden patterns within data (Boutaba et al., 2018).

Moreover, ML operates on the principle of inductive inference, in which a hypothesis (or approximating function) is drawn from past experiences to predict future events (Faceli et al., 2011b). This hypothesis should ideally be as general as possible, as it needs to be valid for unseen objects.

In a broad sense, ML can be studied in three main categories: Supervised Learning (SL), Unsupervised Learning (UL) and Reinforcement Learning (RL). In SL, the labels for the objects used are known, often referred to as target or meta attributes. In contrast, UL involves situations where labels are unknown, and the objective is to explore or describe a dataset (Faceli et al., 2011b). RL is an ML paradigm centered on actions and rewards. This study specifically focuses on SL and RL.

2.3.1 Supervised Learning

Within the field of SL, we can categorize models into two types: classification and regression. In classification problems, the goal is to find a hypothesis that separates the data into discrete value classes. Conversely, in regression problems, the objective is to determine continuous values (Boutaba et al., 2018). In both cases, it is feasible to visually represent the problems using a line (straight line) on a graph. This line can serve as a decision boundary for classification problems or a regression line for regression problems, as illustrated in Fig. 6.

In this Thesis, several ML models were considered, including Decision Tree (DT), Random Forest (RF), K-Nearest Neighbors (KNN), and Neural Networks (NN).

DT is a method based on the divide-and-conquer strategy, aimed at solving a complex problem by decomposing it into smaller subproblems. In this approach, solutions to the subproblems are combined in a tree-like structure, ultimately yielding a solution to the original problem (Faceli et al., 2011b). The goal of the Decision Trees (DT) algorithm is to minimize the sum of squared residuals, as described in Equation 1, where $\hat{y}R_j = \sum i \in R_j \frac{Y_i}{|R_j|}$.

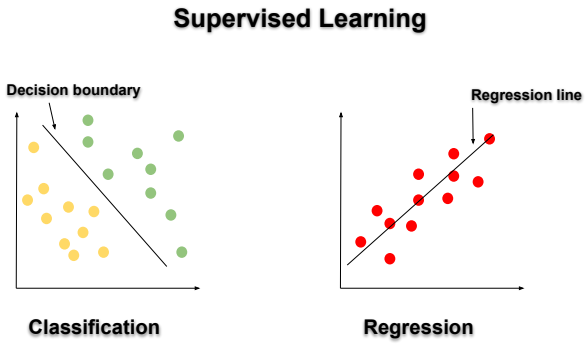


Figure 6 – Types of problem in Supervised Learning.

$$\sum_{j=1}^{J} \sum_{i \in R_k} (y_i - \hat{y}_{R_j})^2 \tag{1}$$

In this scenario, the feature space X with n characteristics is divided into J distinct and non-overlapping regions, R_1, R_2,...,R_j. For each observation falling within a particular region R_j, a prediction is made, which entails taking the average of response values from the training observations within R_j. Additionally, $\hat{y}_{R_j}$ represents the mean response for the $\mid R_j \mid$ observations in the j-th region (Calasans, 2020).

RF is a model that extends DT by employing a strategy that combines multiple trees. The final estimate is computed by averaging the estimates produced by each tree. Each tree is constructed using a random subset of the input attributes X, which introduces variability in the structure of each tree (Stadler; Pasquini; Fodor, 2017).

KNN is a ML model that relies on the Euclidean distance (refer to Equation 2), guided by the intuition that objects associated with the same concept tend to be proximate to one another.

$$dist(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2}$$

In Equation 2, the distance is computed as the sum of differences between vectors x and y, where both x and y are vectors of the same dimension (n).

NN represent a ML methodology designed to replicate the biological architecture of human neurons. These networks comprise densely interconnected processing units, often referred to as artificial neurons (θ), which utilize internal mathematical functions. These units are structured into one or more fully connected layers, where each connection features a weight (w) that adapts during each learning iteration. For example, in Fig. 7, you can observe two artificial neurons in the input layer, one artificial neuron in the hidden layer, and one artificial neuron in the output layer.

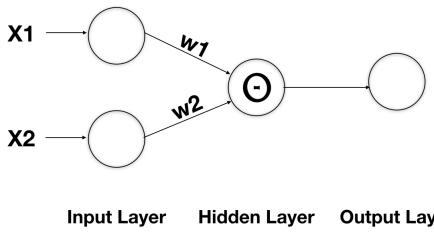


Figure 7 – Neural Network with one hidden layer.

In this example, each neuron in the input layer has a value (x_i); these values are multiplied by the weights (w_i) and then summed. The result (output) of this summation serves as the input for the next layer of neurons, as described by Equation 3.

$$\sum_{i=1}^{n} x_i \times w_i \geq \theta \tag{3}$$

The neural network algorithm encompasses an error correction mechanism that leverages the optimization of a mean squared error function, aligning the network's outputs with the expected values. In this context, artificial neurons are harnessed to make predictive estimations in regression scenarios.

2.3.2 Reinforcement Learning

Reinforcement Learning (RL) is an Artificial Intelligence (AI) paradigm based on actions and rewards. It differs from other approaches most frequently studied in the ML research field, such as supervised and unsupervised learning, since the objective of an RL learner is learning to map situations to actions to maximize a reward signal, instead of mapping the features describing a situation to a label (supervised paradigm) or to find hidden patterns in collections of unlabeled data (unsupervised paradigm). (Boutaba et al., 2018; Sutton; Barto, 2018).

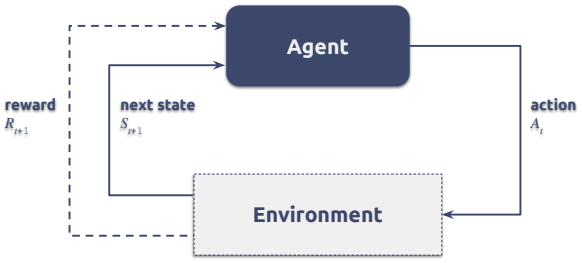


Figure 8 – Interaction between the agent and environment in a MDP. Adapted from (Sutton; Barto, 2018).

The model depicted in Fig. 8 illustrates the formalization of a sequential decision-making strategy known as a Markov Decision Process (MDP). In this framework, the agent continually interacts with the environment by executing actions (A) at specific time steps (t) and observing new states (S_{t+1}) resulting from these actions. After each

interaction, a reward value (R_{t+1}) is generated to evaluate the correctness of the action, to maximize cumulative rewards throughout the agent's training process (Boutaba et al., 2018; Sutton; Barto, 2018).

In this context, the agent learns to maximize its cumulative rewards by determining a policy[2] that optimizes an action-value function, denoted as Q. This function estimates the quality of actions taken by the agent in specific states.

Formally, an optimal action-value function, denoted as q_*, can be defined using the Bellman optimality equation (Sutton; Barto, 2018):

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a] \tag{4}$$

Intuitively, the Bellman optimality equation suggests that the optimal Q-value for any state-action pair ($q_*(s, a)$) is expected to be the immediate reward obtained after taking an action (a) in a given state (s) at time step (t), augmented by the maximum expected return achievable by adhering to an optimal policy for subsequent state-action pairs, which are discounted[3] by γ (Sutton; Barto, 2018).

Hence, the resolution of the Bellman optimality equation provides a pathway to ascertain an optimal policy, offering a potential solution to a RL problem. Nevertheless, it is imperative to acknowledge that, in practice, this solution is seldom feasible. It resembles an exhaustive search that requires consideration of all possible scenarios, involving the computation of occurrence probabilities and expected reward returns. Additionally, it relies on three assumptions that are often challenged when implementing solutions for real-world problems:

a) The accurate knowledge of environmental dynamics.

b) Sufficient computational resources to complete the computational requirements of the solution.

c) The adherence to the Markov property.

In light of these challenges, the only pragmatic approach to tackle the Bellman optimality equation is to seek a policy approximation derived from actual experiences, where transitions involving state s, action a, and reward r are considered, as opposed to relying solely on the expected outcomes (Sutton; Barto, 2018).

When dealing with scenarios characterized by a well-defined set of finite states, it becomes feasible to model an approximation of the Bellman optimality equation using tabular data structures. Each entry within these structures corresponds to a state-action

[2] A policy defines the agent's strategy for associating actions with states. Such strategies can be stochastic, specifying probabilities for each action that could be taken when a particular state is observed (Sutton; Barto, 2018).

[3] This discounting approach allows the agent to prioritize actions that maximize the cumulative rewards it receives in the future. The discount rate, denoted as γ, determines the present value of future rewards. For instance, a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately (Sutton; Barto, 2018).

pair. In this context, the Q-Learning algorithm, introduced by Watkins (Watkins; Dayan, 1992), marked a significant milestone in the early stages of the RL paradigm.

Q-Learning is noteworthy for its direct approximation of the Bellman optimality equation, irrespective of the policy in use. It simplified the analysis of agent algorithms and facilitated early convergence proofs (Sutton; Barto, 2018). The Watkins Q-Learning algorithm is formally defined as follows:

$$Q(S_t, A_t) = (1 - \alpha)Q(S_t, A_t) \ + \ \alpha[R_{t+1} \ + \ \gamma \max_a Q(S_{t+1}, a)] \tag{5}$$

where the approximated optimal Q-value is calculated by blending the current Q-value, denoted as $Q(S_t, A_t)$, with the target temporal difference. The target temporal difference represents the reward R_{t+1} obtained when transitioning to the subsequent state S_{t+1} after taking action a. This value is then weighted by the discount factor γ and modulated by a learning rate α ($0 \leq \alpha < 1$) (Watkins; Dayan, 1992; Sutton; Barto, 2018).

Nonetheless, it should be noted that Q-learning assumes a tabular representation of state-action pairs, whereas real-world applications tend to encompass high-dimensional state spaces unfeasible to be stored in tables, given the impractical computer resources required for such (Sutton; Barto, 2018). In this sense, network monitoring and management applications that utilize a large volume of data, such as fine-grained INT measurements, serve as examples of cases that may be affected by the limitation presented by Q-learning.

To address the Q-learning limitation, (Mnih et al., 2015) leveraged the Q-Learning algorithm by integrating it with a Deep Neural Network (DNN) to approximate the optimal Q-value, a methodology known as DQN. In their work (Mnih et al., 2015), the authors showcased the effectiveness of this approach by training and evaluating the DQN on an Atari 2600 emulator. Impressively, the DQN-based agents achieved performance levels surpassing those of human players in 49 distinct games, relying solely on pixel inputs and game scores for guidance.

Of note, the authors maintained a consistent algorithm, DNN architecture, and hyperparameters across all games, eliminating the need for game-specific feature engineering. Thus, DQN not only outperformed agents employing linear function approximation but also demonstrated the capacity to attain or exceed human-competitive skills across diverse gaming environments. This pioneering work exemplified the synergy between RL and contemporary Deep Learning (DL) techniques, a significant advancement in the field of AI. It underscored the potential of RL when combined with modern DL methods, yielding remarkable outcomes (Mnih et al., 2015; Sutton; Barto, 2018).

2.3.2.1 Deep Q-Network workflow

The DQN architecture, as proposed by Mnih et al. (Mnih et al., 2015), consists of a Deep Convolutional Neural Network (CNN) designed to receive emulated game frames as input and subsequently generate predicted Q-values for each potential action within

the given input state. To facilitate such predictions, Mnih et al. introduced two critical modifications to the conventional Q-Learning algorithm. These alterations were essential to mitigate instabilities inherent in using DNN for Q-value approximation (Sutton; Barto, 2018).

The first modification entails the incorporation of a biologically inspired mechanism referred to as "experience replay". In this approach, the agent's experiences are stored as tuples containing the current state (S_t), the action taken (A_t), the reward received (R_{t+1}), and the subsequent state (S_{t+1}). Periodically, after reaching a predefined replay memory limit, a mini-batch of these experiences is uniformly sampled for training the DNN (Mnih et al., 2015; Sutton; Barto, 2018).

This approach plays a fundamental role in mitigating the emergence of correlations within the observed state space. By decoupling the dependence on successive experiences, it effectively reduces the variance in the parameters of the DNN. Fig. 9 illustrates the interaction between a DQN agent and an environment, taking into account the experience replay mechanism. Within this context, the agent selects actions following an ϵ-greedy rule.

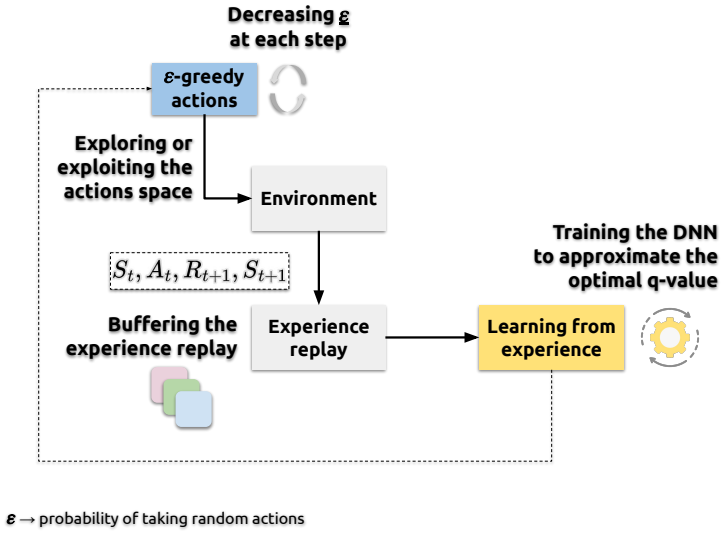


Figure 9 – DQN high-level workflow (Mnih et al., 2015; Sutton; Barto, 2018).

Specifically, when employing this rule, the agent chooses between two strategies: "exploitation" and "exploration". A "greedy action" involves selecting an action from the action space based on the maximum estimated Q-value. Conversely, a "non-greedy action" entails the random selection of an action. Exploitation, represented by the selection of a greedy action, aims to exploit the current knowledge to maximize immediate rewards. In contrast, exploration, represented by non-greedy actions, focuses on traversing the action space to maximize cumulative rewards in the long run (Sutton; Barto, 2018).

In RL, achieving a balanced trade-off between exploration and exploitation is paramount. However, it's important to acknowledge that, at a single time step, it's not possible for an

agent to simultaneously exploit and explore actions. To conciliate these opposing strategies, a solution is to allow the agent to primarily act greedily, favoring exploitation, while intermittently choosing an action from the action space at random, independent of the estimated Q-values. This random selection is determined by an exponentially decreasing probability parameter ϵ. Consequently, as the time steps progress, the probability of selecting an optimal action gradually converges to a value greater than $1 - \epsilon$, approaching near certainty in favor of exploitation as the agent refines its strategy over time (Sutton; Barto, 2018).

A second significant contribution introduced by Mnih et al. (Mnih et al., 2015), relative to classical Q-Learning, pertains to the learning stage of the DQN. In this stage, a separate network, referred to as the "target network", is employed to estimate target values for the Q-network, often referred to as the "online network". This modification enhances the algorithm's stability compared to using a single online network. The rationale behind this improvement lies in the fact that updating the parameters of the online network for the current state-action pair can inadvertently influence the Q-values of the next state, potentially leading to oscillations or even policy divergence.

To address this challenge, the online network's parameters are periodically cloned to the target network at intervals of every C time step. Consequently, the target network's predictions serve as target values for the online network during the subsequent C time steps. This introduces a delay in updating the Q-values between the current and next states, effectively reducing the likelihood of policy oscillations or divergence (Mnih et al., 2015). Figure 10 illustrates the DQN learning workflow, incorporating the approach described above.

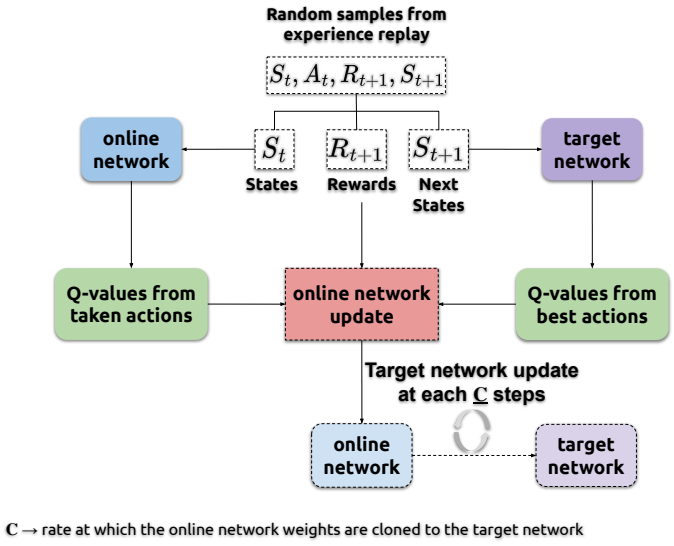


Figure 10 – DQN learning stage workflow (Mnih et al., 2015).

In the context of this thesis, our premise revolved around optimizing the integration of the data generated by INT with ML models. We have harnessed the synergy of these

two tools, using them as a supplementary mechanism for congestion control.

2.4 Congestion Control

Over the past thirty years, the topic of Internet congestion has received widespread attention from the scientific community (Peterson; Brakmo; Davie, 2022). Since the first paper (Jacobson, 1988) published, the topic of Internet congestion has been consistently characterized as a "resource allocation" problem. In the context of this thesis, these "resources" can be understood such as link bandwidth and buffer space. Based on this understanding, congestion control mechanisms (CC) were created to allocate resources fairly within the shared network infrastructure.

To achieve greater fairness in the use of shared network resources, CC mechanisms reduce the transmission of additional data packets within the network. However, depending on the intensity and attributes of the network traffic, the CC mechanisms implemented at the host level may prove insufficient in ensuring optimal network utilization. Such limitations arise when network buffers are full, forcing the router to employ a tail-drop mechanism to discard the most recently received packets. This approach is considered suboptimal due to its propensity to induce "TCP global synchronization" (Malangadan; Raina; Ghosh, 2023).

In this case, the CC necessitates the integration of auxiliary mechanisms to mitigate network congestion, such as AQM, a traditional mechanism employed in network device queues.

2.4.1 Active Queue Management

The main objective of an AQM algorithm is to effectively manage network congestion and prevent queues from reaching their maximum buffer capacity. The primary methods for conveying congestion conditions to senders include packet **marking** using ECN bits and selective packet **dropping**.

Using ECN for packet marking plays a role in congestion control by explicitly indicating to CC mechanisms that they should reduce their transmission rates. We can observe that CC mechanisms take advantage of ECN bit marking covering Data Center TCP (DCTCP) (Alizadeh et al., 2010) and, more recently, TCP Prague (Briscoe et al., 2018).

On the other hand, in the scenario of probabilistic packet discards by an AQM mechanism, CC stands to gain from the process of fast retransmission. In this instance, the sender detects the occurrence of duplicate acknowledgments (typically three in number) and initiates the retransmission of the packet that had been deliberately discarded by the AQM as part of its congestion mitigation strategy within the router's queues (Peterson; Brakmo; Davie, 2022).

Within this thesis, the subject matter of AQM assumes a central and fundamental role in enhancing the QoS for the DASH service. A comprehensive discussion of our proposed solution will be presented in Chapters 5 and 6.

2.5 Summary of the Chapter

In this chapter, we provided a concise overview of essential concepts to facilitate the understanding of Smart Closed Loop. Consequently, we delved into the topics of adaptive video streaming, network programmability, machine learning, and congestion control. Our objective was to enable readers to discern with greater clarity the precise positioning of the concepts that materialize the Smart Closed Loop.

Chapter 3

Related Work

This chapter provides an overview of the most recent related work. In Section 3.1, we delve into the subject of video service estimation. These investigations are situated within the context of the "**M**" and "**A**" components of the MAPE cycle. Section 3.2 is dedicated to a comprehensive examination of research associated with the "**E**" component, specifically focusing on the developments in AQM within the programmable data plane. Lastly, in Section 3.3, we present the latest research findings of the application of ML techniques to enhance the QoS and QoE in the context of DASH services, fully covering the **MAPE** cycle.

3.1 Video Service Estimation

In the study (Stadler; Pasquini; Fodor, 2017), a ML approach was employed, encompassing the integration of metrics extracted per second from a computing cluster. These metrics encompassed various aspects, including CPU, memory, and disk utilization, in addition to metrics from an OpenFlow network. Specifically, they focused on the data volume in terms of bytes and packets transmitted and received. This set of input features was utilized to estimate various video metrics, such as FPS. The outcome of this investigation yielded a Normalized Mean Absolute Error (NMAE) of 9.17% in the FPS prediction. It is important to clarify that the study (Stadler; Pasquini; Fodor, 2017) was primarily founded on the analysis of a static video service, deviating from the dynamicity of the DASH. In contrast, our work successfully tackled this dynamicity and demonstrated a reduction in prediction errors.

In the study (Calasans, 2020), bytes and packets transmitted and received in an OpenFlow setup served as the basis for ML models estimate DASH metrics. It is worth noting

that the video and audio parameters employed in this investigation closely mirrored those employed in this thesis. Notably, despite the exclusive reliance on network-derived metrics, the work (Calasans, 2020) reported a NMAE of 16.5% in their evaluation.

Work	NMAE audio	NMAE video
(Stadler; Pasquini; Fodor, 2017)	20.6%	9.17%
(Calasans, 2020)	13%	16.5%
This thesis (Chapter 4)	**2.80%**	**7.22%**

Table 1 – DASH service metrics estimation.

As briefly described in Table 1 and detailed in Chapter 4, the findings of this thesis supported an improvement in DASH service metric estimation, reaching 2.80% and 7.22% of NMAE error. We posit that the utilization of fine-grained telemetry metrics for estimating video service metrics yields superior outcomes.

Work	Fine-grained (INT)	Timescale	Buffer metrics	Data plane
(Stadler; Pasquini; Fodor, 2017)	x	sec	x	x
(Calasans, 2020)	x	sec	x	x
This thesis (Chapter 4)	✓	μs	✓	✓

Table 2 – Characteristics used to estimate DASH service metrics.

Table 2 enumerates the main attributes that delineate the advancements achieved within this thesis in comparison to existing literature regarding the estimation of DASH service metrics. We can observe that our work utilizes fine-grained measurements of buffer metadata collected directly from the data plane on a microsecond timescale.

3.2 AQM in programmable data planes

Since the paper introducing P4 in 2014 (Bosshart et al., 2014b), the scientific community has shown interest in the implementation of AQM flavors in the programmable data planes. The initial work was published in 2018 (Kundel et al., 2018), titled "P4-CoDel: Active Queue Management in Programmable Data Planes". It presented a P4 implementation of the CoDel (Nichols; Jacobson, 2012) algorithm in a software switch environment.

In the following year (2019) two other works (Menth et al., 2019) and (Papagianni; Schepper, 2019) were published in the same context. The first, published in July 2019, whose title is "*Implementation and Evaluation of Activity-Based Congestion Management Using P4 (P4-ABC)*", presents an AQM prototype in P4 based on the ABC strategy using software switch and externs (out of data plane domain) to workaround the data plane limitations. In December 2019, the second one was published with the title "PI2 for P4: An Active Queue Management Scheme for Programmable Data Planes", representing the first P4 implementation of the Proportional Integral controller in a software switch environment. Part of the AQM logic was implemented in the control plane due to constraints in mathematical operations within the data plane.

It was only in 2021 that a more robust performance analysis of a hardware version of P4-CoDel (Tofino architecture) was presented (Kundel et al., 2021) in the work "*P4-CoDel: Experiences on Programmable Data Plane Hardware*". Still in 2021, the thesis "*Making a Packet-value Based AQM on a Programmable Switch for Resource-sharing and Low Latency*" was defended proposing PV-AQM (Toresson, 2021), an AQM based on Per-Packet Value (PPV) and Proportional Integral Controller Enhanced (PIE) in programmable data plane hardware (Tofino architecture). As in (Papagianni; Schepper, 2019), they also resorted to the control plane to calculate the drop probability with the PIE controller.

In August 2022, a more in-depth evaluation of PI2 in hardware (Tofino architecture) was published (Gombos et al., 2022) in the work "*Active Queue Management on the Tofino programmable switch: The (Dual)PI2 case*". Furthermore, an extension of PI2 (dualPI2) to support the L4S framework was also developed and presented in the same work. In both versions, the control plane has been used to assist in computing complex mathematical operations performed by the PI controller. In the same year, the first version of iRED (**a result of this thesis**) was presented in a software switch environment (v1model) (Almeida et al., 2022), evaluating an adaptive video streaming scenario in the work "*iRED: Improving the DASH QoS by dropping packets in programmable data planes*". In this previous version, iRED obtained superior results about the state-of-the-art AQMs and the Tail Drop approach. Still in 2022, FG-AQM was published (Qiao; Gao, 2022) in the work *"Fine-Grained Active Queue Management in the Data Plane with P4"*. In this case, FG-AQM uses a PI controller to compute the drop probability in a software switch environment.

In 2023, a new implementation of the CoDel algorithm, referred to as CoDel++ (Doan et al., 2023), in a PDP using priority queues was proposed in the work "Interplay Between Priority Queues and Controlled Delay in Programmable Data Planes". In this new version, the authors evaluated CoDel++ on a Tofino hardware switch. In the same year, a new version of iRED with full support for the L4S framework was proposed in (Almeida et al., 2023). This work involved a comprehensive performance evaluation conducted on the Tofino2 hardware.

Work	Delay based	Depth based	Ingress drop	L4S compliance	Fully imp. in the data plane	Hardware version
P4-CoDel	✓	*x*	*x*	*x*	✓	✓
P4-ABC	✓	*x*	✓	*x*	*x*	*x*
PI2 for P4	✓	*x*	*x*	✓	*x*	✓
PV-AQM	✓	*x*	✓	*x*	*x*	✓
FG-AQM	✓	*x*	*x*	*x*	*x*	*x*
CoDel++	✓	*x*	*x*	*x*	✓	✓
This thesis (Chapter 5)	✓	✓	✓	✓	✓	✓

Table 3 – AQM comparative analysis.

Table 3 provides a succinct comparative analysis of AQM implemented within a programmable data plane. The table showcases the advancements achieved by the research

conducted in this thesis, reflecting the state-of-the-art progress in AQM context.

3.3 QoS/QoE improvements in DASH scope

The KNN-Q algorithm, a fusion of KNN and Q-Learning techniques, was introduced in (Lin et al., 2020) as an approach aimed at enhancing the QoE in DASH scope. The simulations demonstrated that the utilization of KNN in conjunction with Q-Learning mitigates the challenges associated with a large state space. In this particular scenario, KNN-Q was integrated within the client-side video player.

In (Wei et al., 2021), a RL framework was introduced to enhance QoE within the context of DASH. In this framework, the video player assumes the role of an agent, while the fluctuations in the network state serve as the environment. Rewards are computed based on the video quality and player buffer status. The primary objective of the agent is to make optimal selections of video segments to maximize the accumulated rewards.

The study presented in (Kim; Chung, 2022) combines the capabilities of Edge computing and RL to enhance the QoE in a DASH service. In this context, a mobile edge computing framework collects measurements from clients, including bandwidth, bitrate, and buffer level. These measurements serve as inputs to the RL algorithm. The RL agent, utilizing this data, dynamically adjusts the bitrate adaptation policy on the client side with the ultimate aim of maximizing the QoE.

The DQNReg, which was introduced in (Hafez; Hassan; Landolsi, 2023), represents a RL strategy designed to improve the QoE in DASH scenarios. This strategy leverages DQN to enhance the adaptive logic algorithm in the video player. Specifically, the input layer of the DQN is constituted by network throughput, segment bitrate, and buffer occupancy of the client video player. The reward function is a composite measure that takes into account the segment bitrate, buffer level, and rebuffering duration using a segment-wise approach.

The common thread among all the related work described in this section is their emphasis on the video client side. The measurements utilized as observed states originate from the video player, and the actions taken are associated with the logic of ABR adaptation. One of the outcomes of this thesis is DESiRED, described in detail in Chapter 6, which adopts a different perspective of the problem. In contrast to the client-side approach, which predominantly focuses on capturing client state information, the DESiRED framework directs its focus toward the monitoring and intervention processes at the network nodes. It leverages fine-grained network telemetry as the input for a DRL mechanism and orchestrates real-time adjustments in the AQM target delay.

In this context, it is crucial to emphasize that the approach proposed in this work is uniquely inclusive, as it focuses its monitoring and interventions exclusively on the network. In essence, it is entirely feasible to integrate DESiRED with any of the approaches

mentioned in this section. Moreover, we anticipate that such integration has the potential to augment the outcomes achieved by these approaches.

Table 4 presents a compilation of the state-of-the-art employing machine learning models to enhance the QoS and QoE in the context of DASH services. We can observe that our proposal stands out for being the only one that addresses adjustments to the network instead of the video player.

Work	Input layer	Agent Action	Reward	ML strategy
(Lin et al., 2020)	coarse-grained (player)	bitrate selection	QoE	KNN-Q
(Wei et al., 2021)	coarse-grained (player)	bitrate selection	QoE	DDPG
(Kim; Chung, 2022)	coarse-grained (edge comp.)	bitrate selection	QoE	DQN
(Hafez; Hassan; Landolsi, 2023)	coarse-grained (player)	bitrate selection	QoE	DQN
This thesis (Chapter 6)	fine-grained (INT)	AQM target delay	QoS	DQN

Table 4 – Main characteristics used to improve DASH QoS/QoE.

3.4 Summary of the Chapter

In this chapter, a comprehensive review of the current state-of-the-art of service estimation, AQM, and QoS improvements were presented. We have delineated the principal characteristics of the relevant works and described how our approach extends beyond the existing research.

Chapter 4

Using ML and INT for Service Metrics Estimation

This chapter presents the challenges and motivations behind the DASH service estimation[1], characterizing the **Monitoring** and **Analysis** components within the MAPE cycle. In this context, we provide a solution to make QoS estimations of a DASH service using In-Band Network Telemetry (INT) to feed Machine Learning (ML) models. Moreover, we provide a comprehensive examination of the strategies employed in the literature, as well as an in-depth exploration of the advancements achieved through the research conducted in this thesis. We still offer a concise summary of our findings and present our concluding remarks.

4.1 Contextualization and Motivation

The extensive array of network metrics made available through INT offers significant benefits for network administrators, as INT empowers them with novel insights into network dynamics, encompassing packet pathways, buffer occupancy, and queue waiting times. This fine-grained data could be a valuable resource for data-hungry ML algo-

[1] This chapter is based on the following works:

- ALMEIDA, Leandro C. de; VERDI, Fábio L.; PASQUINI, Rafael. **Estimando métricas de serviço através de In-band Network Telemetry.** Brazilian Simposium on Computer Networks and Distributed Systems (SBRC 2021)Uberlândia, MG, Brazil.
- ALMEIDA, Leandro C. de; VERDI, Fábio L.; PASQUINI, Rafael. **Using Machine Learning and In-band Network Telemetry for Service Metrics Estimation**. 10th IEEE International Conference on Cloud Networking (IEEE CloudNet 2021), Virtual Conference.

rithms. With this knowledge, they could accurately forecast the operational status of services within the network, thereby creating opportunities for management actions.

In this context, this Chapter intends to answer the following research questions:

1. Can INT metadata be used to feed (as input layer) ML algorithms?
2. Is it possible to obtain a more accurate estimate than that provided in the related work (Stadler; Pasquini; Fodor, 2017; Calasans, 2020)?
3. What is the most appropriate model for estimating DASH QoS metrics?

By leveraging the synergy between INT and ML, the objective of this Chapter is to investigate the potential of their combined application in service metrics estimation. More specifically, our focus is on estimating the QoS for DASH services. It is important to clarify that DASH represents the prevailing technology choice for video streaming, being adopted by industry leaders such as Netflix® and Google® (Lederer, 2015). DASH's unique offering lies in its adaptive rate video service, allowing clients to adjust various parameters, including resolution, bitrate, and frames per second, in response to network load conditions (ISO, 2014). The estimation of metrics for an adaptive service like DASH is inherently challenging due to the dynamic nature of network load conditions, which constantly fluctuate.

4.2 Problem Statement

In this chapter, we initially operate under the assumption of a Content Delivery Network (CDN) environment. Within this context, the estimations generated by the ML can prove valuable for Network Service Providers in adapting their infrastructure in response to the outputs of the estimation. Furthermore, we hypothesize that fluctuations in service metrics have a positive correlation with network infrastructure measurements.

In our investigation, we conducted an in-depth analysis to examine the correlations between INT metrics, referred to as data set X, and service metrics, designated as data set Y. In essence, the fine-grained network metrics in dataset X represent the network state, while the dataset Y encompass data sourced from the video components, specifically focusing on FPS.

Following the approach presented in (Stadler; Pasquini; Fodor, 2017), we adopt the use of a global clock, which is accessible to all network devices, ensuring precise synchronization across all devices. Furthermore, we treat the progression of metrics X and Y as time series denoted as X_t, Y_t, and (X_t, Y_t) , where each time instant t signifies the network and video service's state.

The central challenge in our study lies in estimating the service metrics Y_t at time t, based on the detailed network telemetry metrics X_t. This problem can be formally

represented as $M : X_t \rightarrow \hat{Y}_t$, where $\hat{Y}_t$ serves as an approximation of the function Y_t for a given X_t. In essence, this constitutes a regression problem, typically addressable through supervised ML techniques, as discussed in the domain of statistical learning (James et al., 2013).

4.3 Evaluation

In this section, we offer a concise overview of the evaluation for DASH service estimation. For a comprehensive and detailed examination of the implementation, we kindly direct the reader to Appendix B.

In our investigation, we utilized well-established ML algorithms to identify the most suitable regressor for DASH service estimation, using INT measurements as the input layer. Specifically, we evaluated the performance of the Decision Tree (DT), Random Forest (RF), K-Nearest Neighbors (KNN), and Deep Neural Networks (NN). In this evaluation, we designed a dataset containing a diverse range of traffic patterns managed by Workload Assay for Verified Experiments (WAVE) (Almeida et al., 2023), including sinusoidal, flashcrowd, and a simultaneous combination of both. These traffic patterns were intentionally chosen to provide a model with a robust training experience, encompassing transitions from low to high network load conditions. Additionally, we introduced random microbursts into the dataset, creating a diverse load profile with variations in flow characteristics, such as size (long and short), application types (web services, management), and duration times.

Initially, we aim to determine the most important[2] feature among the evaluated ML algorithms. Fig. 11 shows which features (INT metadata) contributed the most for the learning in all load conditions. In this case, on the x-axis, we have the Gini importance (Nembrini; König; Wright, 2018) representing the impurity level of the feature, and in the y-axis we have the name of the feature (metadata).

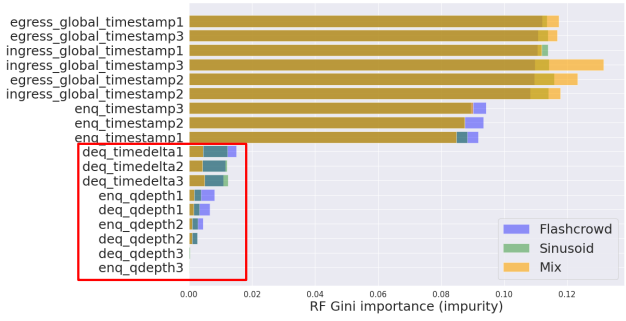


Figure 11 – Random Forest feature importance.

[2] The feature that has more impact in the training phase.

The lower the level of impurity, the better the feature is classified. Therefore, we can observe the metadata related to buffers (highlighted with a box) has the most influence on the learning model. Such a finding is of great importance since it gives some hints that not all the network metrics may need to be collected. It points to a direction where the ML algorithms may have good accuracy with less data.

Still in Figure 11, we can observe that the node closest to the client (See topology in Appendix B) has more influence on the learning than all the others. We conjecture that the intermediate nodes represent a good point of view of the state of the network, however, it is the closest node to the client that has the most accurate information for the client state. This result supports a similar finding in related work (Stadler; Pasquini; Fodor, 2017).

Furthermore, we evaluated what is the best ML model in terms of NMAE, discussing the ability of ML models to predict QoS metrics from different load conditions. In this sense, the results in Table 5 were obtained with the best estimator after the grid search cross-validation and hyperparameters optimization for each load pattern.

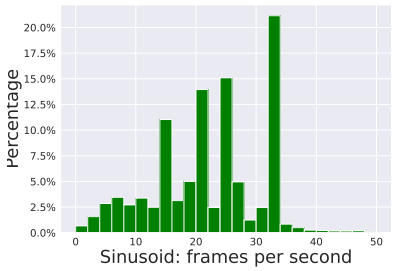
Load	DT	RF	KNN	NN
Sinusoid	33.74%	7.23%	25.81%	26.85%
Flashcrowd	30.28%	6.70%	20.54%	19.94%
Mix	25.71%	2.96%	29.82%	21.42%

Table 5 – NMAE for load patterns individually.

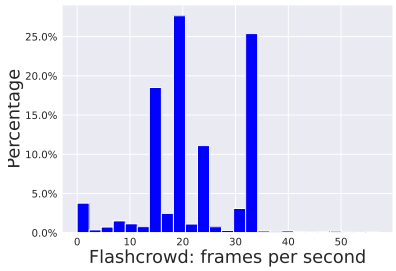
Table 5 shows that RF obtained the lowest NMAE at evaluating individual datasets, that is, the model was trained, tested and evaluated in every individual load pattern. A NMAE of **7.23%** was obtained for the sinusoid load, **6.70%** for the flashcrowd load and **2.96%** for the mix load, having a mean NMAE of **5.63%**. Unexpectedly, the RF achieved its best performance in the mix load pattern, which is the worst scenario considering that both loads (sinusoid and flashcrowd) run at the same time. Although this pattern of traffic is challenging for the ML algorithms, it mimics the real traffic load. We believe that this behavior was because the mix pattern generated a high load, and made few transitions in the network when compared to the sinusoid and flashcrowd patterns. This can be inferred by observing histograms in Figure 12, in which the x-axis represents the number of the FPS played in the client and the y-axis the percentage of this FPS frequency in the experiment.

When looking at the sinusoid load (Figure 12(a)), we observe that the client played the video with a high resolution (around 30 FPS) in 21% of the time and made transitions to other minor resolutions (around 18 and 24 FPS). Figure 12(b) (flashcrowd load) shows that the client played the video in the lower resolution (around 18 FPS) in 27% of the time. Transitions to higher resolutions were done at other moments in time. It is important to say that about 4% of the time the client couldn't play the video. In the mix load (Figure 12(c)), the client played the video in high resolutions (around 30 FPS) in just 11% of the time. This indicates that the load in the network was higher for a longer time. This

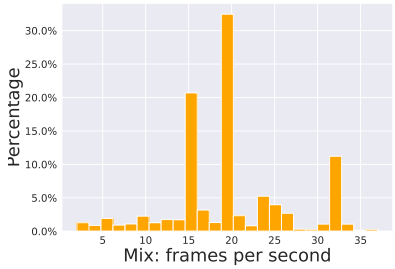
high load caused a lower number of transitions in the resolutions when compared to the other two experiments, which is easier for ML algorithms' estimations, leading to a better NMAE.



(a) Sinusoid



(b) Flashcrowd



(c) Mix

Figure 12 – Load Patterns.

Utilizing the best regressor, which in this case was the RF, we depict a visual representation of the estimated values compared to the real values for video metrics. Specifically, Fig. 13 provides a graphical representation of the estimated values concerning the real values for the video metric (FPS).

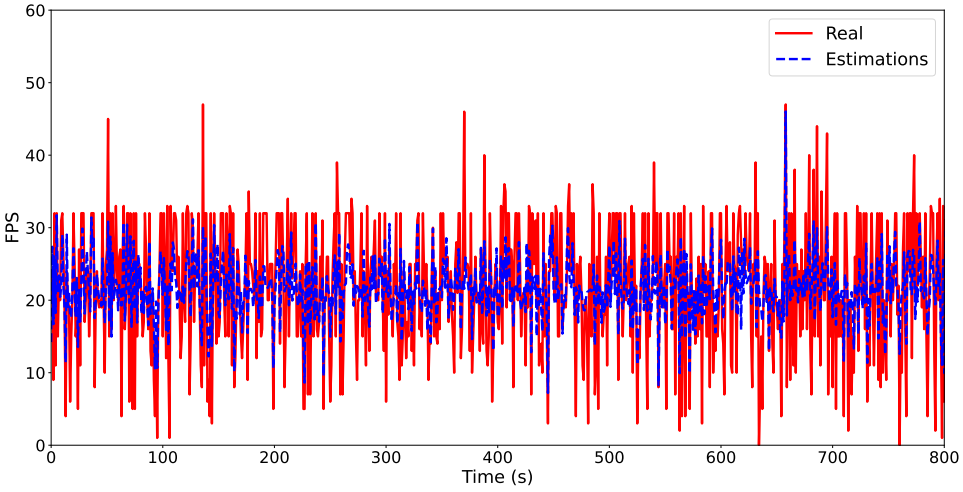


Figure 13 – Real and estimation computed by RF regressor.

The estimation curve depicted in Fig. 13 shows a tendency to mirror the behavior of the reality curve. However, due to its regression nature, the estimates are susceptible to

abrupt changes in real FPS values. In such instances, employing a continuous learning strategy, such as reinforcement learning, could be considered a viable option.

Finally, we evaluated the ability of a ML model, trained in one load pattern, to predict service metrics from other load patterns, as described below and shown in Table 6.

- ❏ Sinusoid → Flashcrowd: the model was trained with the sinusoid load and evaluated with the flashcrowd load.
- ❏ Sinusoid → Mix: the model was trained with the sinusoid load and evaluated with the mix load.
- ❏ Flashcrowd → Sinusoid: the model was trained with the flashcrowd load and evaluated with the sinusoid load.
- ❏ Flashcrowd → Mix: the model was trained with the flashcrowd load and evaluated with the mix load.
- ❏ Mix → Sinusoid: the model was trained with the mix load and evaluated with the sinusoid load.
- ❏ Mix → Flashcrowd: the model was trained with the mix load and evaluated with the flashcrowd load.

Load	DT	RF	KNN	NN
Sinusoid → Flashcrowd	30.89%	33.22%	33.88%	35.69%
Sinusoid → Mix	28.75%	30.98%	41.00%	36.33%
Flashcrowd → Sinusoid	33.72%	36.15%	36.38%	42.69%
Flashcrowd → Mix	30.04%	30.34%	40.64%	43.57%
Mix → Sinusoid	35.11%	38.15%	37.42%	40.26%
Mix → Flashcrowd	31.11%	34.30%	34.03%	37.77%

Table 6 – NMAE for cross evaluations.

The general conclusion is that the models perform poorly when evaluating against other load patterns. It is conjectured that this happens because the load patterns explored different regions of the space of possibilities (system states), as seen in the histograms of Figure 12.

4.4 Summary of the Chapter

This chapter presented a step forward toward the estimation of DASH QoS metrics using INT and ML, covering the **Monitoring** and **Analysis** components within the MAPE cycle. We answer the three questions listed at the beginning of this chapter as follows:

1. It is possible to use measurements from telemetry to estimate service metrics;

2. It is possible to obtain results with a lower NMAE error in service estimates;

3. The random forest demonstrated to be the model with the best performance, that is the lowest error.

A network service provider could leverage the methodologies and insights discussed here to realize tangible benefits, including the implementation of real-time, fine-grained monitoring for DASH flows transiting its network. Within the Smart Closed Loop, the solution proposed in this chapter serves as a trigger for potential action strategies, aiming to maintain an adequate level of QoS. The next chapter discusses one of these strategies, which is based on a probabilistic packet discard policy.

Chapter 5

Mitigating Network Congestion through Probabilistic Packet Dropping

Once the network service provider is aware of the video quality through the monitoring and analysis strategies discussed in the previous chapter, we can evaluate solutions to maintain an adequate level of QoS for a DASH client. Among the various possible solutions, we chose to use a policy based on packet discarding, as DASH uses TCP as a transport protocol, which reacts to the non-receipt of confirmations. In this scope, this chapter covers the **Execution** component of the MAPE cycle, being dedicated to a comprehensive exploration of Active Queue Management (AQM) within the context of programmable data planes (PDP). We start this topic by delineating a challenge encountered in the context of AQM when deployed within a PDP, commonly referred to as the "Egress Drop Problem". Following, we provide a brief introduction to the L4S framework, representing one of the most recent initiatives by the IETF aimed at addressing congestion issues on the Internet. The main point of this chapter lies in the comprehensive exposition of iRED[1], a disaggregated P4-AQM solution, which is entirely realized within

[1] This chapter is based on the following works:

- ALMEIDA, Leandro C. de; MATOS, Guilherme; PASQUINI, Rafael; PAPAGIANNI, Chrysa, VERDI, Fábio L. **iRED: Improving the DASH QoS by dropping packets in programmable data planes.** 18th International Conference on Network and Service Management (CNSM). November 2022, Thessaloniki, Greece.
- ALMEIDA, Leandro C. de; MATOS, Guilherme; PASQUINI, Rafael; PAPAGIANNI, Chrysa, VERDI, Fábio L. **Improving the DASH QoS by dropping packets in programmable data planes.** IEEE Global Communications Conference (IEEE Globecom), DEMO Session, Rio de Janeiro - Brazil, 2022.

the data plane, supports the L4S framework and saves router resources.

5.1 The Egress Drop Problem

In this section, we elucidate the operation of a typical PDP switch, providing insights into the precise conditions that give rise to the Egress drop problem – where, how, and why it occurs. Additionally, we explain the advantages of separating the decision-making process from the packet-discarding action within an AQM logic. To comprehensively understand the origins of this issue, we delve deeper into the architecture of a standard programmable switch in Fig. 14.

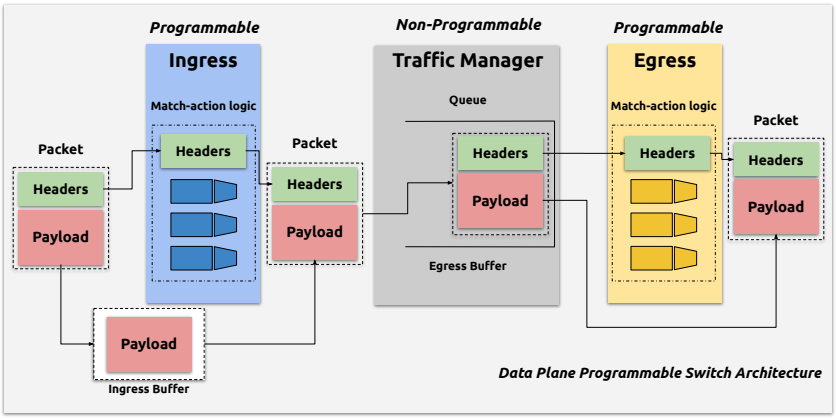


Figure 14 – The generic architecture of the data plane programmable switch. Headers and Payload follow different paths on the device.

As detailed in Fig. 14, a generic switch architecture with a PDP is composed of some programmable blocks (Ingress and Egress) and non-programmable components (Traffic Manager). After the packet is received by a given ingress port, it is separated into Headers and Payload. Headers are the structures that are actually processed by programmable blocks. It is from the data contained in the header fields and other metadata that the programmer can define logic based on match-action to accomplish what is desired with the network packet. On the other hand, the Payload remains unchanged, usually stored in buffers, throughout the packet processing. After processing the Ingress block, the packet is reconstructed, generally by a Deparser (omitted in the Figure), which unifies the header with the payload that was stored in the Ingress buffer. The packet is then sent entirely to the Traffic Manager, which positions the entire structure in a queue associated with an output port. After the packet is serviced by the scheduler, it is separated again so that the Headers can be processed by the Egress block. As with the Ingress buffer, the payload remains in the Egress buffer unchanged. After the Header passes through the

❏ ALMEIDA, Leandro C. de; PASQUINI, Rafael; PAPAGIANNI, Chrysa, VERDI, Fábio L. **iRED: A disaggregated P4-AQM fully implemented in programmable data plane hardware.** Preprint 2023. (TNSM under review).

necessary stages in the Egress block, the packet is then reassembled to be forwarded or marked for drop, if applicable.

As already mentioned, the most important data for the AQM logic is the queuing delay (or queue depth) which is only available at the Egress block. The way the AQM works is by setting a FLAG which informs to the Egress block that the packet must be dropped. This action is performed only after the end of Header processing in the Egress block, that is, the buffer resources (memory) that are being used by the Payload are finally released. In this work, we define this waste of resources as **The Egress drop problem**.

Understanding the causes and effects that this problem brings, we argue that it is possible to improve the use of shared resources (switch pipeline). The idea we defend is that the **decision** of the drop must be separated (in different blocks) from the **action** of the actual drop, thus having a disaggregated concept of AQM. The materialization of this new concept is described in Section 5.3, in which we present the iRED algorithm.

5.2 L4S - Low Latency, Low Loss, and Scalable throughput

As briefly mentioned previously, the L4S architecture (shown in Fig. 15) introduces incremental changes to both the hosts' CC algorithm and the AQM at the network nodes. The modifications proposed by L4S were motivated by some requirements, such as L4S-ECN packet identification, accurate ECN feedback, fall-back to Reno-friendly on Loss, fall-back to Reno-friendly on classic ECN bottleneck, reduce RTT dependence, scale down to the fractional window and detecting loss in units of time (Briscoe et al., 2018).

In this context, L4S introduces two distributed mechanisms that work together to achieve the requirements listed above. The first of these reside in the host scope, being the scalable CC algorithm, TCP Prague[2](Briscoe et al., 2018). The TCP Prague is a modified version of DCTCP (Alizadeh et al., 2010) for safe use over the Internet. As is well-known (by TCP researchers) DCTCP is suitable only for data centers, where the administrator can arrange the network to work properly for frequent ECN-marking. However, this is not so simple for the public Internet, as DCTCP flows would certainly starve classical flows. For this reason, TCP Prague presents minor modifications from DCTCP to meet the requirements listed above.

The second resides in the network nodes as a Dual-Queue coupled AQM (Schepper; Briscoe; White, 2023), that is responsible for maintaining a harmonious coexistence between the flavors of CC, Classic and Scalable. The Dual Queue coupled AQM mechanism, specified in the RFC9332 (Schepper; Briscoe; White, 2023), was designed to solve the coexistence problem, accommodating flows into separated queues for Classic (larger

[2] The name is after an ad hoc meeting of IETF in Prague in July 2015.

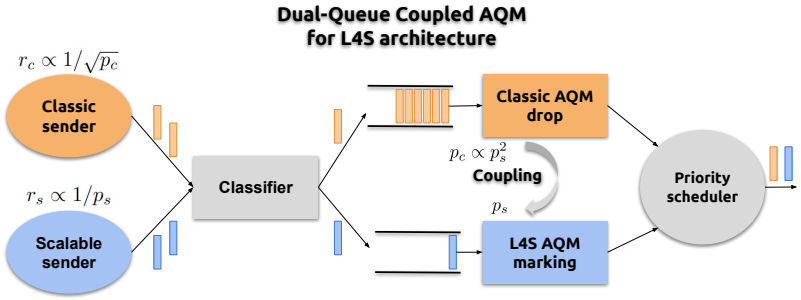


Figure 15 – Dual-Queue AQM in L4S architecture. Adapted from (Schepper; Briscoe; White, 2023).

queueing delay) and Scalable (small queueing delay) CC flavors, as can be seen in Fig. 15.

Despite the use of distinct queues with varying depths (shallow and deeper), bandwidth consumption remains uniform across flows. Achieving equitable resource allocation, or harmonious coexistence, involves the interplay between the Classic and Scalable queues. This interaction enables the Classic queue to perceive the square of congestion levels in the Scalable queue. This squared is then offset by the sending rate of the classic sender (r_c) in response to a congestion signal, characterized by $r_c \propto 1/\sqrt{p_c}$, where p_c denotes the loss level of the Classic flow. On the other hand, the Scalable sender rate (r_s) follows an inverse linear approach, characterized by $r_s \propto 1/p_s$, where p_s denotes the loss level of Scalable flow. It is this linearity that characterizes scalability in response to congestion. In the next section, we will describe iRED, our AQM proposal that supports the L4S framework.

5.3 iRED - ingress Random Early Detection

iRED was designed under three fundamental premises: *i*) Perform probabilistic packet dropping with minimal overhead; *ii*) Support and adhere to current Internet congestion control mechanisms, such as the L4S framework; *iii*) Be fully implemented in the data plane hardware. Based on these guiding requirements, this Section describes the details and challenges of implementing iRED on the Tofino2 programmable switch[3].

Regarding the first premise, we understand that to minimize overhead on the switch, iRED should be able to discard packets as soon as possible. Leveraging the programmable switch pipeline, we believe that the most suitable place to perform the drop action is in the Ingress block. However, the data (queue metadata) necessary to calculate the drop probability is available after the Traffic Manager, that is, in the Egress block. In this context, we decided to divide iRED's operation into two parts, making it a disaggregated

[3] The previous version of iRED (Almeida et al., 2022) was deployed in a software switch environment.

AQM. As can be seen in Fig. 16, **decisions are made at the Egress, while actions are performed at the Ingress.**

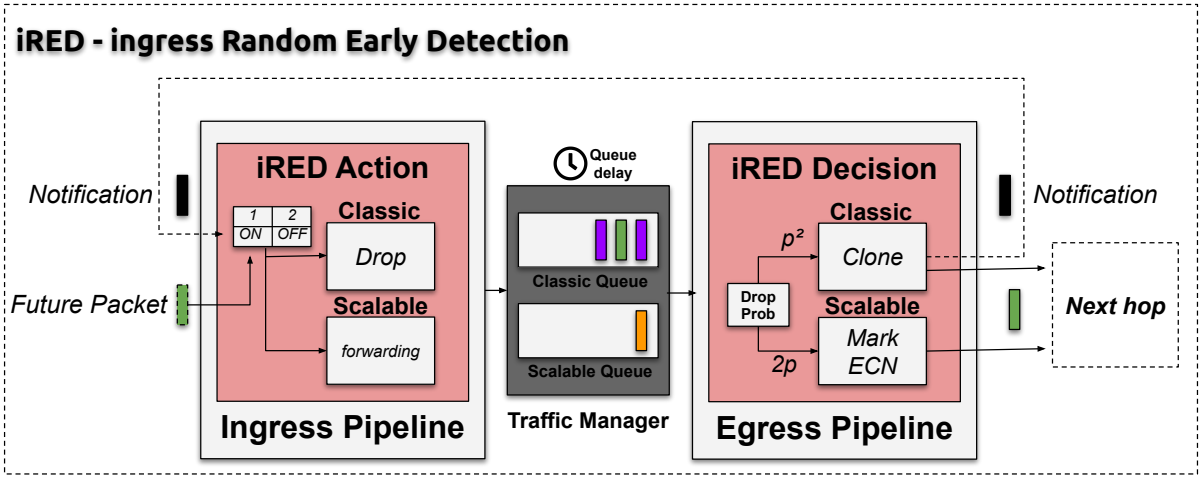


Figure 16 – iRED design. Disaggregating the action of a drop decision reduces wasted resources.

In alignment with the second premise, we implemented the AQM requirements presented previously in Sec. 5.2 to provide support for L4S. First of all, the classification process is performed in the Ingress block, in which the logic identifies the type of flow and enqueues it to the corresponding output queue. Furthermore, the coupling mechanism is implemented in the Egress block. In this scenario, iRED dynamically adjusts the drop probability or marking based on the flow type (Classic or Scalable).

Finally, iRED is fully implemented in the hardware improving the autonomy in performing AQM functions only within the data plane, thereby eliminating the control plane or external mechanisms to make specific tasks. In this context, it is well-established that AQM logic requires the utilization of intricate mathematical operations, including multiplications, divisions, and square roots. Furthermore, certain sections of the logic require the implementation of more sophisticated functions, such as exponential moving averages or similar calculations. We overcome the challenges imposed by the architecture and implement iRED entirely in the data plane using available resources, such as bitshift to represent mathematical operations and compute the Exponentially Weighted Mean Average (EWMA).

For a more comprehensive understanding, we will initiate the description of iRED's operation from the Egress block, specifically commencing with the drop or mark decision (decision module). At the Egress, iRED computes the EWMA of the queue delay (or queue depth[4]) for each individual packet, entirely within the data plane. The inherent absence of division and floating-point operations poses challenges in calculating average values within the data plane. To overcome this limitation, as applied in (Busse-Grawitz et al., 2022), we employ an approximation method following Eq. 6:

[4] The programmer can choose whether to use iRED's delay-based or depth-based approach.

$$S_t = \alpha \cdot Y_t + (1 - \alpha) \cdot S_{t-1} \tag{6}$$

where S_t is the updated average queue delay, S_{t-1} is the previous average queue delay and Y_t is the current queue delay. The constant $\alpha \in [0, 1]$ determines how much the current value influences the average. We use $\alpha = 0.5$, such multiplication can be replaced by bit shifts operations. The output of the EWMA will represent the average queue delay over time. If the value observed (average queue delay) is between a set of min-max thresholds defined, iRED will compute the drop probability according to the RED approach and will, based on the coupling mechanism, generate different congestion signal intensities (drop or marking).

Once the iRED decision module (Egress) has detected that a packet must be dropped (Classic), iRED must notify the action module (Ingress) to perform this action. The first challenge in the PDP context is to achieve communication between the Ingress and Egress blocks with minimum overhead. Obviously, iRED will not drop the packet that generated the discard decision, but a future packet (Chen et al., 2019). Discarding future packets is one of the main features differentiating iRED from other state-of-the-art AQMs. For the congestion notification to reach the Ingress block, iRED creates a congestion notification packet (clone packet with only 48 bytes) and sends it through an internal recirculation port to reach the Ingress block.

Algorithm 1 presents the iRED decision module, which operates within the Egress block. This module continuously monitors the queue delay (or depth) and maintains an updating register that stores the probability for dropping (Classic) or marking the ECN bit (Scalable).

Algorithm 1 functions as follows: when the packet is identified as a clone, it is recirculated to the Ingress block (lines 5-6). This action signifies that future packets should be dropped in the Ingress for the designated output port, thereby consuming only 48 bytes per packet. For regular packets, not cloned, the current queue delay is employed to calculate the EWMA based on Equation 6 (line 8). If the EWMA value falls within the defined minimum and maximum thresholds (line 9), iRED proceeds to calculate the probability of dropping or marking with ECN. The decision module employs a random number generator to compute distinct probabilities for each traffic type (lines 10-11). It is noteworthy to clarify that for L4S packets, the marking probability is twice as high as that for classic packets (coupling mechanism). Consequently, the random number used in the computation of the L4S marking probability is half of the random number employed for determining the drop probability, as defined by the L4S framework (Briscoe et al., 2023).

The subsequent step in the algorithm involves identifying the packet type, which could be L4S or Classic. If the packet type is determined to be L4S (line 12), the decision module proceeds to compare the randomly generated L4S number with the drop probability value

Algorithm 1 Decision to drop - Egress

```
1: minThsld = TARGET_DELAY|QUEUE_DEPTH
2: maxThsld = 2 * minThsld
3: dropProb = 0
4: for each pkt do
5:     if pkt == pktCloned then
6:         recirculate(pkt)
7:     else
8:         EWMA = 0.5 · queue_delay + (1 − 0.5) · EWMA_{t−1}
9:         if (EWMA ≥ minThsld) and (EWMA ≤ maxThsld) then
10:            randClassic = random(0, 65535)
11:            randL4S = randClassic/2
12:            if L4S then
13:                if randL4S < dropProb then
14:                    markECN(pkt)
15:                    dropProb = dropProb − 1
16:                else
17:                    dropProb = dropProb + 1
18:                end if
19:            else
20:                if randClassic < dropProb then
21:                    dropProb = dropProb − 1
22:                    clone(pkt)
23:                else
24:                    dropProb = dropProb + 1
25:                end if
26:            end if
27:        end if
28:        if EWMA > maxThsld then
29:            if L4S then
30:                markECN(pkt)
31:            else
32:                clone(pkt)
33:            end if
34:        end if
35:    end if
36: end for
```

stored in a register (line 13). If this comparison yields a true result, indicating that the L4S packet should be marked, the ECN bit of the L4S packet is set to 1 (line 14), and the drop probability value stored in the register is decremented by one unit (line 15). Conversely, if the condition is false (line 16), the drop probability value in the register is incremented by one unit (line 17).

For Classic traffic, the logic is analogous (lines 20-24). However, instead of marking the ECN bit, the decision module executes a clone operation (line 22). In the clone operation, the original packet remains unaltered and proceeds to be forwarded as usual to its final destination. Simultaneously, the clone packet is modified to carry only notification information destined for the action module.

In cases where the EWMA value exceeds the established maximum threshold (line 28), a uniform action is taken: either all packets are marked as L4S or all packets are cloned as Classic, depending on the traffic type (lines 29-32).

The action module, situated in the Ingress block, maintains the congestion state table on a per-port/queue basis and activates the drop flag (ON) for the corresponding port/queue. The current packet is forwarded to the next hop without introducing any additional delay. Subsequently, future packets intended for the same output port/queue,

where the drop flag is set to ON, will be dropped (classic), and the drop flag will be reset to OFF. This mechanism, facilitated by iRED, ensures that the Ingress pipeline can proactively mitigate imminent queue congestion.

Now, we will explain the action part of iRED (listed in Algorithm 2), which runs at the Ingress block. The initial step in Algorithm 2 involves verifying whether the incoming packet has been recirculated from the Egress block (line 2). We employ a register with a length matching the number of ports, where each port is associated with an index. If the packet has been recirculated, the drop flag is activated by setting the corresponding value in the index register to 1 (line 3). Following this, the recirculated packet serves its purpose and is subsequently discarded (line 4).

Algorithm 2 Action to drop - Ingress

Input: *pkt, pktRecirc*

```
1:  for each pkt do
2:      if pkt == pktRecirc then
3:          dropFlag[output_port] = 1 {Flag to drop ON}
4:          drop(pktRecirc)
5:      end if
6:      ip_forward
7:      dropPort = dropFlag[output_port]
8:      if dropPort == 1 then
9:          if L4S then
10:             forwarding
11:         else
12:             drop(pkt) {Packet dropped}
13:             dropFlag[output_port] = 0 {Flag to drop OFF}
14:         end if
15:     end if
16: end for
```

The remainder of Algorithm 2 primarily focuses on the routine forwarding of packets (line 6), where the output port is determined. In this step, the algorithm evaluates to ascertain the status of the drop flag associated with the specified output port. Should the flag be in the activated state (indicated by a value of 1), the packet undergoes a dropping procedure (as delineated in lines 12-13), and concurrently, the register is restored to its initial state. It is important to note that only one packet is dropped at a time, and subsequent packets destined for the same output port will only be dropped if a recirculated packet is detected in the Ingress pipeline, signaling congestion.

For Scalable flows, iRED does not drop packets, as expected; instead it forwards the packet to the scalable queue. In summary, iRED is the only current AQM P4-based that drops packets in the Ingress block, fully deployable in the programmable data plane hardware and is L4S-capable.

5.4 Evaluation

In this evaluation, we have conducted three types of experiments. Firstly, we evaluate resource utilization within the context of existing AQM algorithms implemented in the

P4, utilizing a Tofino2 programmable switch as the experimental platform. Following this, we proceed to assess the compatibility of AQM with the L4S framework, with a specific focus on fairness in the concurrent operation of classic (TCP cubic) and scalable (TCP Prague) flow types. Lastly, we evaluate the AQM algorithms within the context of an adaptive video streaming scenario, specifically focusing on DASH.

5.4.1 Resource Consumption Analysis - Tofino2

The objective of this evaluation is to analyze the consumption of switch resources for *all packets discarded by the AQM methods*, that is, the resources that were wasted. In this context, we evaluated four metrics: wasted memory, wasted time, wasted clock cycles (latency), and wasted weight (power consumption). The wasted memory is the sum of all memory resources used by the packets until being dropped, expressed in megabyte (MB). The wasted time is the sum of all time used by the packets until being dropped, expressed in milliseconds. The wasted cycles is the number of clock cycles and weight is a metric that represents the power consumption (unit-less). A comprehensive exposition of the results is provided in Appendix C. The subsequent paragraphs present a consolidated summary of the results.

We performed an in-depth evaluation with the state-of-the-art Egress-based AQMs (P4-CoDel and PI2 for P4) and iRED, reproducing the same setup (traffic intensity) conducted in (Gombos et al., 2022), as detailed in Fig. 17. Our testbed consists of a P4 programmable switch (Edgecore DCS810 - Tofino2), which connects two Linux hosts, Sender and Receiver, having 25Gbps of link capacity. We conducted our experiments over different network conditions shown in Table 7, varying bandwidth, RTT and MTU.

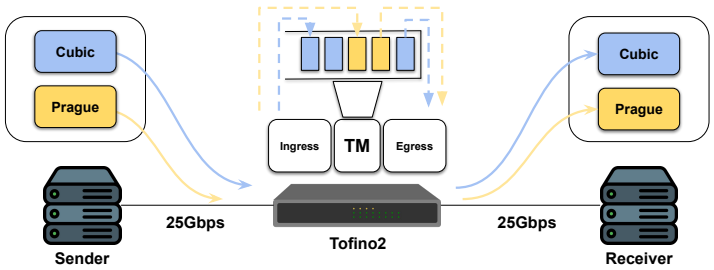


Figure 17 – Evaluation setup. Cubic and Prague flows coexist in the same scenario, sharing the programmable switch bandwidth.

Figs. 18 and 19 show the consolidated overview of the resources saved by iRED for all configurations evaluated. Regarding PI2 (Fig. 18), iRED saves up to 5.6x power consumption, 5.47x clock cycles and 4.77x memory. However, in three configurations (IV, VIII and XII) in which the RTT is 50ms, PI2 wastes fewer resources. We observed that the target delay was rarely reached in these configurations, resulting in few actions of the PI controller.

Table 7 – Configurations

Configuration	Bandwidth(Mbps)	RTT(ms)	MTU(Bytes)
I	120	10	1500
II	120	50	1500
III	1000	10	1500
IV	1000	50	1500
V	120	10	800
VI	120	50	800
VII	1000	10	800
VIII	1000	50	800
IX	120	10	400
X	120	50	400
XI	1000	10	400
XII	1000	50	400

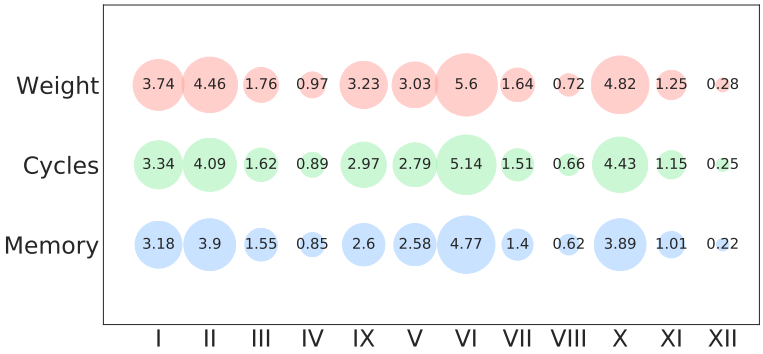


Figure 18 – iRED resource savings compared to PI2.

Regarding P4-CoDel (Fig. 19), the AQM algorithm drops all packets that reached the target delay. Even for the scalable traffic (TCP Prague), all packets are dropped (instead of being marked), since P4-CoDel does not support the L4S. This explains the large number of wasted resources. iRED saves up to 8.9x weight, 12.79x clock cycles and 10.21x memory.

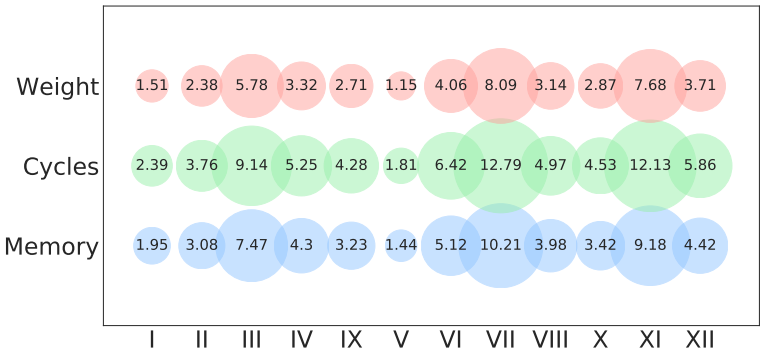


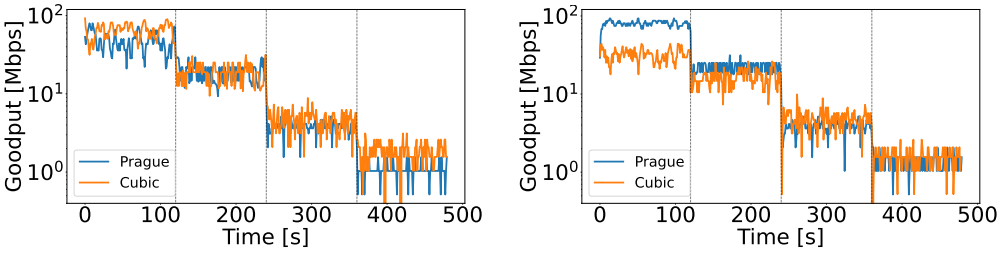
Figure 19 – iRED resource savings compared to P4-CoDel.

5.4.2 Fair sharing in L4S scope

In this particular experiment, our primary objective is to evaluate the support and adherence to the L4S framework. As only iRED and PI2 meet this requirement, P4-CoDel was not evaluated in this experiment. Additionally, we aim to evaluate the harmonious coexistence between non-L4S flows, conventionally referred to as classic (TCP Cubic), and L4S-compliant flows, denoted as Scalable (TCP Prague). We used the same setup as the previous experiment (See Fig. 17), selecting configurations with an MTU of 1500 bytes (I, II, III, and IV).

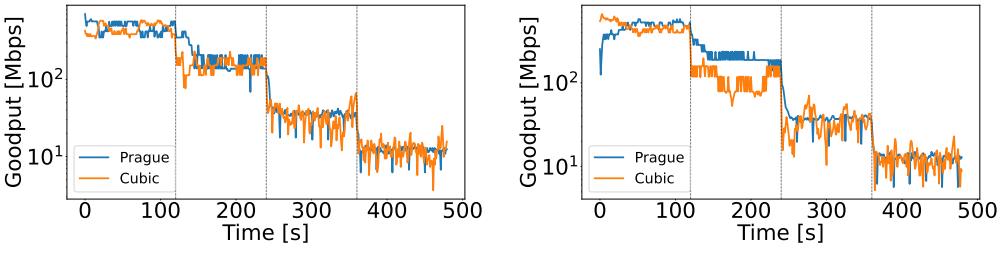
In alignment with the methodology outlined by (Gombos et al., 2022), our experimental configuration involved the imposition of traffic intensity loads comprising four discrete phases, each spanning a 120-second duration. Within each of these phases, we introduced new flows with specific flow pairs (1-1, 2-2, 10-10, 25-25) into the system. This sequential introduction of flows allowed us to initiate the load with lower intensity and progress toward a high-load scenario.

In the context of a 10ms baseline RTT and a bandwidth set at 120 Mbps, Figs. 20(a) and 20(b), becomes evident that a more equitable coexistence between flows is achieved with the implementation of the iRED. Conversely, flows employing the PI2 exhibit a relative disadvantage, with improved fairness only becoming apparent in the latter half of the experiment, specifically during phases 3 and 4.



(a) iRED 120Mbps, 10ms

(b) PI2 120Mbps, 10ms



(c) iRED 1Gbps, 10ms

(d) PI2 1Gbps, 10ms

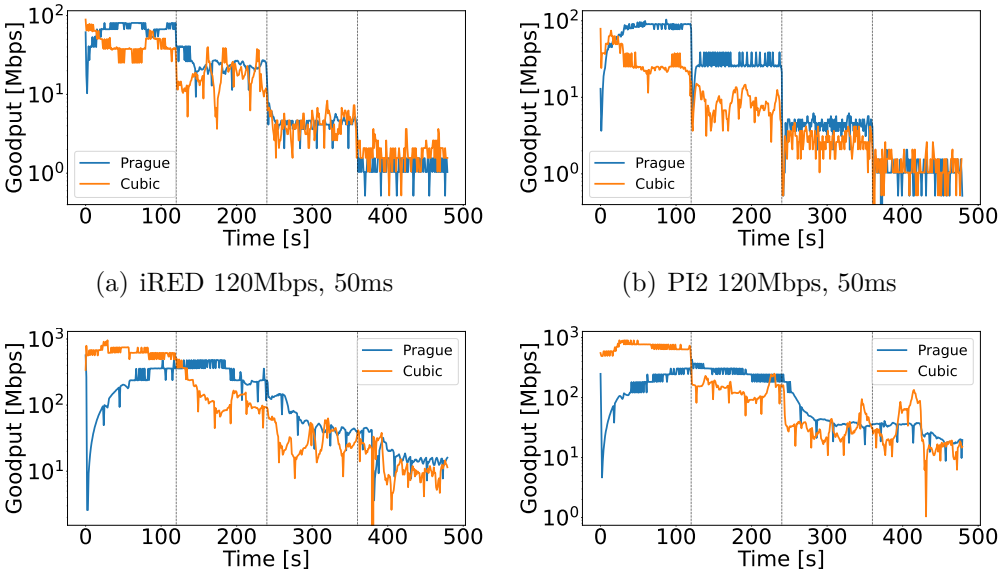
Figure 20 – Coexistence evaluation of Cubic and Prague flows (RTT base 10ms).

When we examine the evaluation outcomes for a 1 Gbps bandwidth and a base RTT of 10 ms, it remains evident that the equitable distribution of shared bandwidth among flows persists across all phases of the experiment when utilizing iRED, as can be seen in Fig. 20(c). In the case of PI2, Fig. 20(d) despite the initial appearance of fairness in the

coexistence of flows during the initial phase of the experiment, this equilibrium does not endure into phase 2.

In Fig. 20, the overarching conclusion drawn from our analysis suggests that in the case of PI2, the intensity (i.e., the probability of marking the ECN bit) required to mark packets from the Prague flow is insufficient during the initial phases of the experiment. This deficiency in marking intensity becomes apparent because the Prague flow, due to its bandwidth consumption characteristics, tends to dominate and not facilitate a fair coexistence with the Cubic flow.

In Figure 21, we evaluate scenarios in which the baseline RTT is configured to 50 ms, a value commonly encountered in long-distance networks. With a bandwidth of 120 Mbps and an RTT of 50 ms, the observed outcomes closely parallel those obtained with an RTT of 10 ms. Specifically, the iRED continues to exhibit superior fairness in the coexistence of Cubic and Prague flows, as seen in Fig. 21(a), while the PI2 attains fairness only in the later stages of the experiment, as can be seen in Fig. 21(b).



(a) iRED 120Mbps, 50ms

(b) PI2 120Mbps, 50ms

(c) iRED 1Gbps, 50ms

(d) PI2 1Gbps, 50ms

Figure 21 – Coexistence evaluation of Cubic and Prague flows (RTT base 50ms)

However, in the case of 1 Gbps and an RTT of 50 ms, both approaches exhibited a parallel pattern of behavior, as can be seen in Figs. 21(c) and 21(d). There was a notable reduction in the performance of the Prague flow during the initial phase of the experiment, followed by a more equitable coexistence between flows in the subsequent three phases. In this particular scenario, we conjecture that the delayed feedback (ACK) to the Prague TCP flow resulted in a slower initial ramp-up, as Prague TCP is notably more dependent on this metric (Briscoe et al., 2018). This sensitivity likely contributed to the observed behavior where Prague TCP experienced a significant drop in performance during the initial phase of the experiment.

5.4.3 DASH scenario

Finally, there is nothing more important than evaluating novel mechanisms using real scenarios and applications. In this experiment, we elucidate the functioning of delay-based AQM mechanisms, specifically P4-CoDel and PI2, with an iRED depth-based version. We employ a DASH test case for our investigation, where the experimental setup comprises three Linux hosts: a DASH server, a video client, and a load generator. These hosts are interconnected via a Tofino 2 switch offering a throughput capacity of 25 Gbps, as depicted in Figure 22.

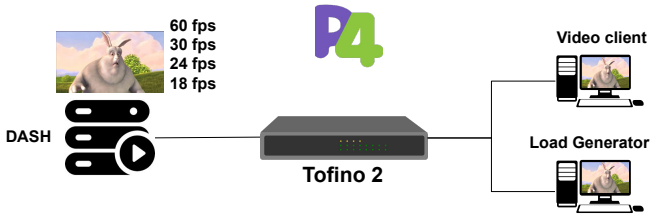


Figure 22 – DASH experiment.

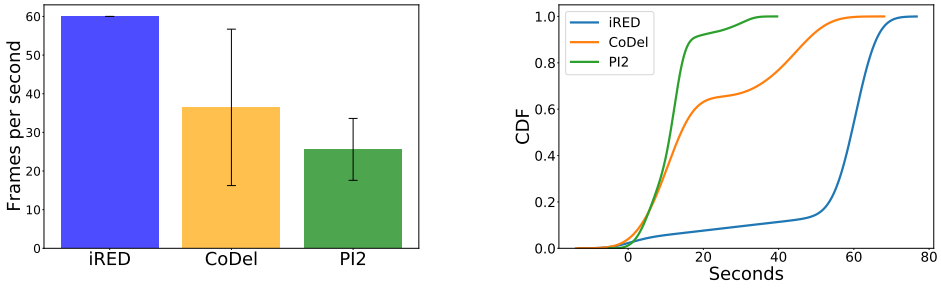
The DASH server houses the Big Buck Bunny video, available in four different quality levels: 60, 30, 24, and 18 frames per second (FPS). The video client can dynamically select from these quality levels based on the network conditions. In instances of elevated congestion, the client opts for lower-resolution video playback. Conversely, during periods of reduced network load, the client selects the highest available video resolution. This dynamic traffic behavior is influenced by the sinusoidal load applied to the testbed, wherein the number of video clients concurrently consuming the video varies between 100 and 150 instances.

The load generator makes requests according to a Poisson process, and the arrival rate is modulated by a sinusoidal function as defined in Equation 7. In this equation, A denotes the amplitude, F represents the frequency, and λ signifies the phase, measured in radians.

$$f(y) = A\sin(F + \lambda) \tag{7}$$

The video client and load generator share the same output queue in the switch. We set the bandwidth (using port shaping) to 100 Mbps as it is the global average broadband speed (Cisco, 2021).

We conducted an evaluation of various AQM strategies, including iRED, P4-CoDel, and PI2, encompassing measurements at application levels. We examined the FPS rendered by the video client and the size of the local buffer employed for storing and playing the next video frames.



(a) FPS (b) Buffer

Figure 23 – DASH Results. iRED improves the DASH Quality of Service.

Figure 23(a) displays the FPS average achieved by the video client for each AQM approach, while Figure 23(b) presents the cumulative distribution function (CDF) of the remaining buffer duration (in seconds) within the video player. It is evident from the results that iRED optimizes both FPS and the available time in the local buffer for video playback. In light of these findings, it is noteworthy that iRED outperforms P4-CoDel by a factor of 1.64x and PI2 by a factor of 2.34x in terms of maximizing FPS. Regarding the video player buffer, our evaluation shows that iRED allows a filling up to 2.57x compared to P4-CoDel and 4.77x compared to PI2.

Our understanding is that iRED has an advantage for latency-sensitive applications due to packet drops in the Ingress block, which minimizes the waste of switch resources. This is in contrast to the TNA implementations of PI2 and P4-CoDel, where packet drops occur on the egress, potentially resulting in less efficient resource utilization. Additionally, the mechanism for discarding packets in the future has a dual impact. First, it ensures that packets experiencing delay are not immediately discarded but are forwarded to their final destination (the video client). Second, this introduces a subtle delay in signaling congestion at the sender. This delay helps to further smooth out TCP's bursty traffic patterns, making iRED particularly effective in maintaining network stability and reducing congestion-induced fluctuations.

5.5 Summary of the Chapter

In this chapter, we cover the **Execution** component within the MAPE cycle, employing probabilistic packet discarding as an approach for AQM to manage the congestion. We presented iRED as a viable solution, offering insights into its operational complexities, efficient utilization of router resources, adherence to the L4S framework, and its significant ability to improve the quality of DASH services. However, iRED suffers from a common issue with all AQM strategies discussed in this work. The fixed target delay problem is detailed in the next chapter, along with our solution to solve it.

Chapter 6

Materializing a Smart Closed Loop Utilizing DRL and INT

As discussed in the previous chapter, iRED uses a fixed target delay (or queue depth) to calculate the drop probability. It would be interesting to have a solution in which the target delay could be dynamically adjusted to better accommodate the DASH traffic behavior. This is precisely what we cover in this chapter. We initiate this chapter with an exposition of a frequently encountered challenge within AQM algorithms, herein referred to as the "fixed target delay problem". After describing the problem, we will present DESiRED[1], our solution for a fixed target delay, leveraging from In-band Network Telemetry (INT) and Deep Reinforcement Learning (DRL) to dynamically adjust the iRED target delay. DESiRED uses the QoS metrics to calculate the rewards or punishments of the smart agent, which can come directly from the video player or be estimated using a regressor, as discussed in Chapter 4. The ultimate goal is to utilize this DRL model for real-time improvements of DASH QoS.

We undertake a comprehensive evaluation of DESiRED within a realistic testbed environment, focusing on the delivery of an DASH service. In this evaluation, we explore the flexibility of DESiRED in two different scenarios: *i*) we use the QoS metrics originating from the video player [2] to calculate the rewards of the RL agent; *ii*) we use the supervised

[1] This chapter is based on the following work:

- ALMEIDA, Leandro C. de; SILVA, Washington Rodrigo Dias da; TAVARES, Thiago C.; PASQUINI, Rafael; PAPAGIANNI, Chrysa, VERDI, Fábio L. **DESiRED - Dynamic, Enhanced, and Smart iRED: A P4-AQM with Deep Reinforcement Learning and In-band Network Telemetry.** Preprint 2023. (Computer Networks - major review).

[2] Although valid in some contexts (eg. network slicing), this scenario does not apply to network service

learning to estimate the state of the video client and use these QoS estimations as input to the RL agent.

Our findings show that DESiRED can reduce video stall occurrences by a factor of up to 90 times. Moreover, the enhancement in the DASH QoS is evident, as measured by an augmentation of up to 42x in terms of FPS, underscoring the considerable efficacy of DESiRED in elevating the QoE.

6.1 The Fixed Target Delay Problem

In the modern domain of computer networks, the necessity to meet rigorous service requirements, including ultra-reliable low-latency communications and high bandwidth, has resulted in a substantial increase in network traffic, amplifying the challenges associated with traffic management. In this sense, AQM approaches that assist congestion control (CC) mechanisms are welcome.

In scenarios where incoming packet rates exceed a network device's processing capacity, transient congestion occurs in the appropriate output queue, often causing transmission delays. To mitigate this bottleneck, an effective strategy involves notification congestion status to the packet sender, allowing the CC algorithm to reduce transmission rates.

Traditionally, AQM mechanisms have been primarily focused on draining packets directly from queues, to mitigate transient congestion and reducing the queue delay. Examples of these traditional AQM algorithms include Random Early Detection (RED) (Floyd; Jacobson, 1993), Blue (Feng et al., 2002), CoDel (Nichols; Jacobson, 2012), CAKE (Høiland-Jørgensen; Täht; Morton, 2018), and PIE (Pan; Natarajan; Baker, 2015). More recently, due to the inherent flexibility of the PDP, the state-of-the-art AQM solutions designed to operate within PDP hardware environments and made publicly accessible comprise iRED (Almeida et al., 2022), P4-CoDel (Kundel et al., 2021), and the (dual) PI2 (Gombos et al., 2022). These AQM implementations exemplify the synergy between novel programmable data plane capabilities and the evolving demands of CC within modern network infrastructures.

An integral aspect of AQM algorithms pertains to setting an appropriate threshold value, often determined based on considerations of queue delay (referred to as the target delay) or queue depth. Opting for a minimal threshold value can lead to an increased occurrence of packet losses, resulting in a higher drop probability while reducing overall link utilization. Conversely, employing a high threshold value can lead to extended queuing delays but a lower likelihood of packet drops, characterized by a reduced drop probability. Additionally, the dynamic nature of network traffic necessitates the avoidance of static threshold values for specific applications. In this context, we explore this issue as the **fixed target delay problem**, as illustrated in Fig. 24.

provider's context, as they do not have access to customer devices.

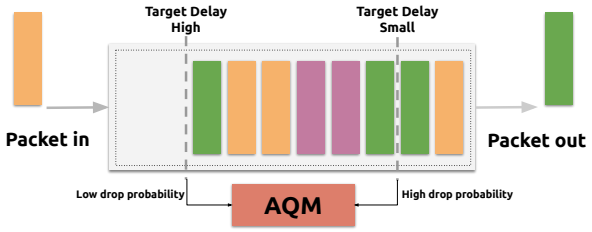


Figure 24 – The trade-off: If the Target Delay is small, it can increase packet losses and decrease link utilization. If it is high, increases queueing delays and decreases packet drops.

Based on this trade-off, it arises a fundamental question: *What is the ideal target delay for AQM?* Estimating this value presents a challenging task. However, recent advancements in the field of AI as applied to computer networks (Boutaba et al., 2018) introduce a potential avenue, leveraging the capabilities of DRL as a powerful tool.

Although DRL models require a high volume of data, providing real-time data at the requisite granularity has posed an obstacle within the context of computer networks. However, recent progress in PDP and INT (P4, 2021), have conducted a paradigm shift. These advancements have given us the capability to attain granular visibility, discernible on a per-packet basis, effectively altering the scenario of the challenges associated with data availability in the context of DRL applications within computer networks.

6.2 DESiRED - Dynamic, Enhanced and Smart iRED

DESiRED, Dynamic, Enhanced and Smart iRED, introduces an innovation where the intelligent control plane leverages the power of DRL to dynamically optimize and adjust iRED target delay parameters on the fly. On the data plane side, DESiRED inherits all the capabilities and functions of iRED. In short, DESiRED materializes a Smart Control Loop.

As discussed in Chapter 5, the concept of disaggregation in the context of this work deals with the ability of AQM to operate on different programmable blocks within the architecture of a PDP. As illustrated in Fig. 25, the decision-making process within DESiRED takes place at the Egress block, while the corresponding actions are subsequently executed at the Ingress block. With DESiRED, the control and data planes interact to materialize an intelligent monitoring and actuation mechanism that aims to maximize the QoS of a DASH service.

The following subsections will describe the functioning of DESiRED, with a distinct focus on data plane and control plane operations.

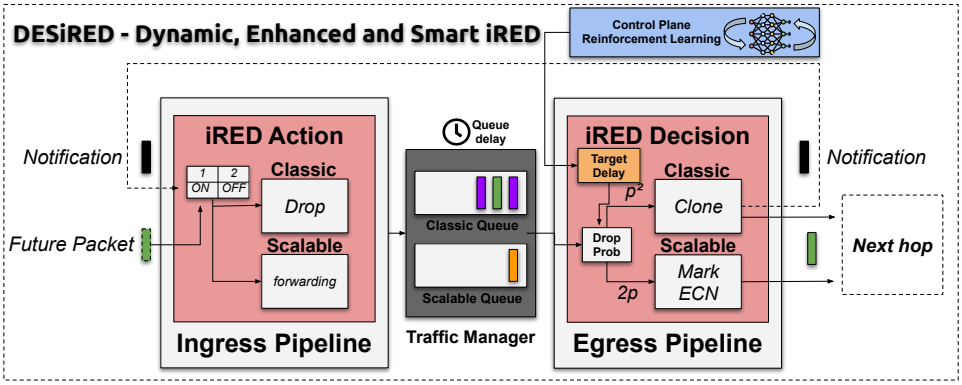


Figure 25 – The Smart Closed Loop overview with DESiRED. At the control plane side, DRL updates the target delay at the data plane.

6.2.1 Data plane operation

The operation on the DESiRED in the data plane is "almost" identical to the operation of the iRED (See Chapter 5). The main difference is that, with DESiRED, the target delay value (threshold) is dynamic. In this case, a specific register (small orange box in Fig. 25) in the Egress Pipeline stores the value of the target delay computed by the intelligent control plane (blue rectangle in Fig. 25). For each packet that enters the Egress, DESiRED reads this value to evaluate whether the average queue delay has reached the defined threshold. The decision to discard (Classic flows) or mark (Scalable flows) packets is carried out in the same manner as iRED.

6.2.2 Control plane operation

As mentioned earlier, DESiRED tackles the issue of fixed target delay through the implementation of an intelligent control plane, denoted by the blue rectangle in Figure 25. This intelligent control mechanism is responsible for updating the register that maintains the dynamic target delay threshold, as determined by the DRL decision process. Now, let's present a comprehensive overview of the control plane operation.

The control plane operates by receiving data from two sources: the **network state** and the **application state**. In this particular implementation, fine-grained INT measurements constitute the input layer for the DQN from the network state. The DQN's output layer is responsible for generating the agent's actions. Concurrently, the application state encompasses DASH metrics, including parameters such as FPS and Local Buffer Occupancy (LBO). The values can be obtained through two approaches: *i*) using the real values from the video player; and *ii*) using the predicted values from the supervised learning component defined in Chapter 4, as can be seen in Fig. 26.

INT measurements comprise observations that depict the network's state with fine-grained detail, offering an unprecedented perspective on the extent of congestion. These measurements are acquired within the PDP and subsequently routed to the intelligent

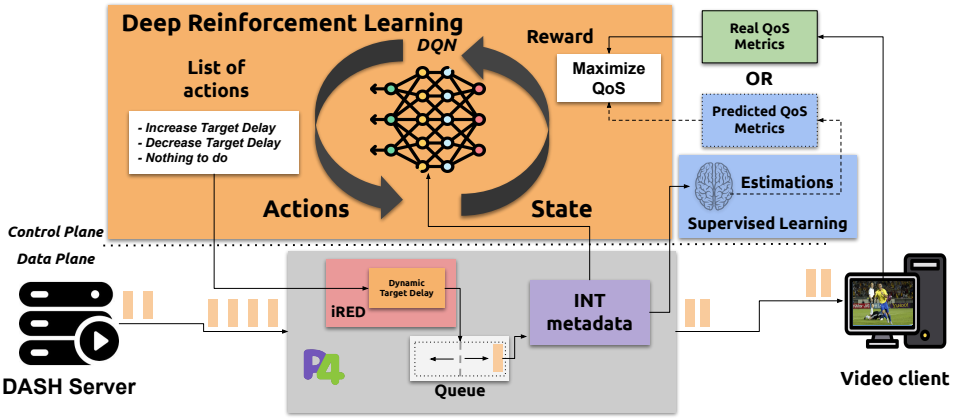


Figure 26 – Operation of the Control Plane in DESiRED involves using fine-grained INT measurements as the input layer for the DQN. Additionally, DASH QoS measurements serve as the basis for calculating agent rewards.

control plane. Within the control plane, they are aggregated into compact data frames, which collectively form what we term the "observation space". In the context of this study, the term observation space refers to the temporal window within which the intelligent control plane conducts an integrated analysis of both the network's state and the application's behavior.

For each received observation space, the DQN incorporates INT measurements as an input layer. Following neural network processing (refining its internal weights), the DQN generates an action, which is manifested as an activation in one of the neurons within the output layer. In this study, the possible actions include:

1. Increase the target delay;
2. Decrease the target delay;
3. Maintain the current state (i.e., take no action).

Subsequently, the control plane retains a record of the executed action and enters a state of anticipation for the next observation space. Upon the arrival of data from the subsequent observation space, the DRL mechanism evaluates whether the undertaken action has led to the optimization of DASH QoS, particularly about enhancements in FPS and LBO metrics. In the event of a positive outcome, the agent is rewarded, whereas in cases of QoS deterioration, the agent incurs a penalty.

Leveraging insights from the dynamic network traffic patterns, DESiRED demonstrates a capability to adapt with precision to prevailing congestion conditions. This adaptability facilitates a continuous enhancement in the quality of video services offered. It is imperative to clarify that DESiRED can be inherently application-agnostic, signifying its capacity to accommodate diverse reward policies tailored to evaluate a wide array of service metrics. This flexibility extends to metrics such as the response time of a web

server or even the frame rate in video playback, underscoring its versatility across various service domains.

6.3 Evaluation

In this section, we provide a summary of the evaluation's key details. For a more comprehensive and in-depth exploration of the implementation, please refer to Appendix D. We undertake a comprehensive evaluation of DESiRED within a realistic testbed environment, using Bmv2 software switch, focusing on the delivery of an DASH service (ISO, 2014) based on two approaches. In the first approach (Case 1), we use real data from the video player to compute the agent's reward policy. In the second approach (Case 2), we predict these data using a trained regressor. All artifacts are available in the public repository [3].

We understand that the first approach (Case 1) may be more suitable for cases where the video service provider has access to network infrastructure metrics. In such scenarios, the content provider may also own the communication infrastructure, thereby allowing unrestricted access to monitor the state of the network.

On the other hand, in the second approach (Case 2), we believe it is more suitable for cases where the network provider does not have access to the video player's metrics. In this scenario, the network provider needs to somehow estimate the QoS of the video player. In this case, the estimates will be provided by the trained regressor, as discussed in Chapter 4 and detailed in (Almeida; Verdi; Pasquini, 2021) and (Almeida; Pasquini; Verdi, 2021).

The setup of these evaluations is depicted in Fig. 27.

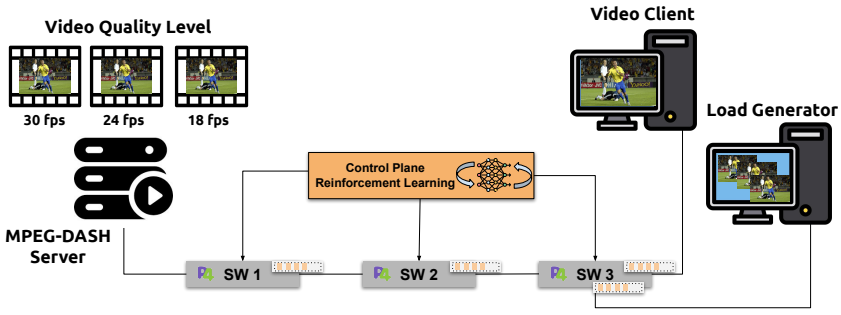


Figure 27 – Setup of DESiRED Evaluation.

6.3.1 Case 1 - Using data from the real video player

Our objective is to evaluate if DESiRED can optimize the QoS for DASH services by dynamically adapting the target delay under conditions characterized by both stationary

[3] https://github.com/dcomp-leris/DESiRED.

and non-stationary loads within a CDN environment. For the baseline, various fixed target delays (5ms, 20ms, 50ms, and 100ms) were implemented within the iRED framework.

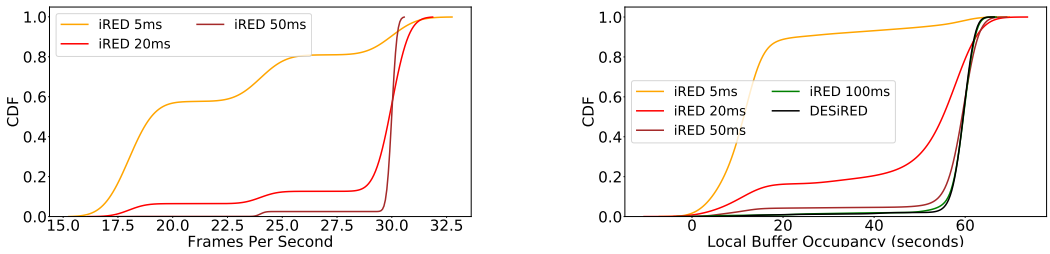
Our experiments involve the provision of diverse video catalogs to video clients traversing a programmable network. Fine-grained INT measurements, collected at line rate in the data plane, are utilized to inform the DRL mechanism in the control plane. The DRL mechanism guides the agent's actions, dynamically adjusting the target delay to optimize the QoS for the DASH service.

We will present the outcomes of our experiments, where we evaluate how DESiRED enhances the QoS of DASH. We offer an in-depth analysis from the client-side perspective, showcasing the results and delving into instances where video QoS has benefited from the dynamic adjustments facilitated by DESiRED. Furthermore, we observe the performance of the DRL model, providing evidence that the agent has successfully learned the designated policy. It has also been able to identify an optimal target delay value that maximizes QoS across the range of experiments conducted.

6.3.1.1 Stationary Loads

The motivation behind evaluating performance under stationary loads stemmed from the necessity to ascertain whether the DRL agent would exhibit distinct learning behaviors during moments of low load (free resources) and high load (congested resources) across separate executions.

When the network load predominantly remains low, network resources are readily available. In such scenarios, there is minimal contention for the use of the queue, resulting in limited or no intervention from auxiliary congestion control mechanisms like AQM. This phenomenon can be observed from the perspective of the video client, as depicted in Figure 28.



(a) FPS (b) LBO

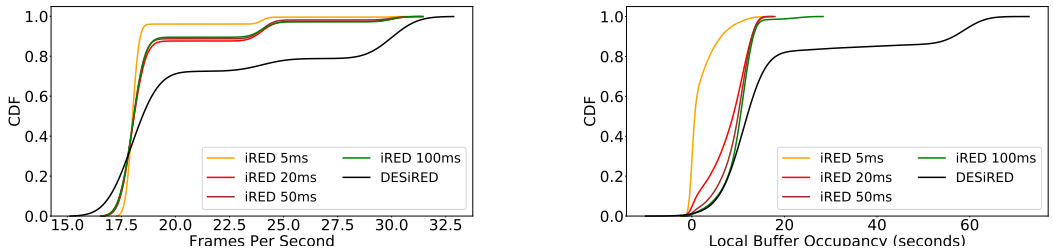
Figure 28 – Low Load - Characterized by only ten video player instances managed by WAVE (Load Generator).

Figures 28(a) and 28(b) illustrate the Cumulative Distribution Function (CDF) of FPS and LBO under low load conditions. In Figure 28(a), we observe some variation in FPS for iRED with a fixed target delay of 5ms and 20ms/50ms. Conversely, in the cases

of iRED with a fixed target delay of 100ms and DESiRED, the video client consistently played the video at 30 FPS throughout all experiments.

Concerning LBO, as depicted in Figure 28(b), the results exhibit similar behavior across approaches, with the local buffer maintaining a near-full state for most of the evaluations, approximately 60 seconds. The only exception is the iRED with a 5ms fixed target delay. In this specific scenario, the use of such a small threshold value appears to have triggered a higher frequency of AQM actions. This, in turn, might have led to more frequent drops within a time interval of less than one RTT, as discussed in (Gettys, 2011). Paradoxically, this increased AQM activity, rather than alleviating congestion, may have exacerbated the situation, demonstrating the potential for unintended side effects when setting overly aggressive congestion control thresholds.

Conversely, when the network experiences predominantly high load conditions, the dynamics shift significantly. In such scenarios, all approaches employing fixed target delay mechanisms encounter challenges in maintaining acceptable DASH QoS. DESiRED, on the other hand, manages to distinguish itself from the fixed target delay approaches, as evident in Figure 29.



(a) FPS (b) LBO

Figure 29 – High Load. Characterized by forty video player instances managed by WAVE (Load Generator).

To gain a deeper understanding of these results, it's important to clarify some aspects of the ABR adaptation logic employed by the DASH.js player, as described in (Spiteri; Sitaraman; Sparacio, 2018). The adaptation logic used in DASH.js, known as DYNAMIC, employs two different algorithms at different stages of video playback. During instances when buffer levels (LBO) are low, such as startup and seek events, a straightforward THROUGHPUT algorithm (based on throughput) is utilized. Conversely, when buffer levels are high, the player switches to the BOLA algorithm (Spiteri; Urgaonkar; Sitaraman, 2020). This dynamic adaptation approach aims to optimize video streaming under varying network conditions, aligning the bitrate selection algorithm with the network's congestion state.

DYNAMIC starts with THROUGHPUT until the buffer level reaches 10s or more. From this point on, DYNAMIC switches to BOLA which chooses a bitrate at least as high as the bitrate chosen by THROUGHPUT. DYNAMIC switches back to THROUGHPUT

when the buffer level falls below 10s and BOLA chooses a bitrate lower than THROUGHPUT (Spiteri; Sitaraman; Sparacio, 2018).

Indeed, from the perspective of the video player's adaptation logic, the LBO metric proves to be far more sensitive to variations in network buffer levels compared to FPS. It's important to note that changes in bitrate and FPS should only occur when the LBO drops below 10 seconds. Consequently, it is logical to aim for maintaining an LBO greater than 10 seconds for the majority of the time, as this instructs the ABR algorithm to select the highest-quality video levels.

Figure 29(b), which pertains to LBO, contributes significantly to understanding why DESiRED achieves superior FPS levels, as indicated in Figure 29(a). In this context, it is plausible to think that fine-tuning the target delay has provided an advantage in terms of preserving a sufficient LBO during periods of severe network congestion. This, in turn, assists the ABR algorithm in making optimal bitrate and quality level selections, ultimately leading to an improved video QoS

6.3.1.2 Non-stationary Load

Recognizing the dynamic nature of network traffic, we evaluated it under non-stationary load conditions. To achieve this, we leveraged the WAVE framework, which effectively managed the execution of video client instances over time, adhering to a mathematical model of sinusoidal periodic load as detailed in Subsection D.0.3 of the Appendix D.

The choice of a sinusoidal periodic load model holds significance because it encapsulates moments of congestion and resource relief in the network, particularly within router buffers, within a single execution. This approach allows us to evaluate our agent's performance in situations of high congestion, where rapid adaptation is crucial, and congestion-free states where shared resources are not overwhelmed. In essence, we expect that the agent will learn distinct patterns that differentiate between these varying states.

This evaluation under non-stationary load conditions provides valuable insights into how the agent responds to fluctuations in network congestion, thereby contributing to a more comprehensive understanding of its adaptability and effectiveness.

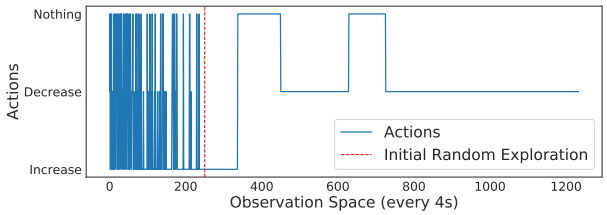


Figure 30 – Actions performed by the agent in the environment. After the initial random exploration, the agent finds the best target delay value to maximize the QoS of MPEG-DASH.

The initial result we would like to present pertains to the actions taken by the agent

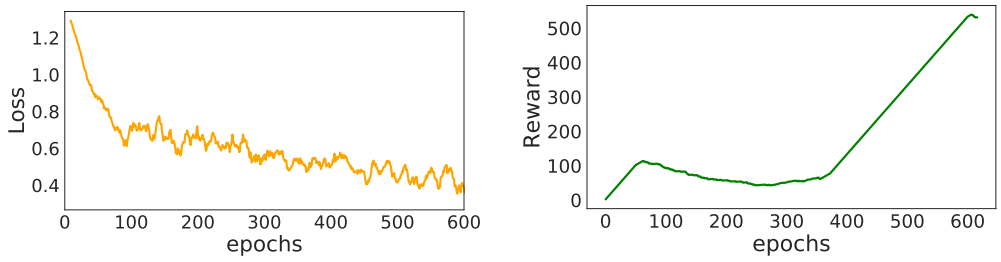
(DESiRED) within the network environment. Figure 30 provides an overview of the agent's actions throughout the experiment. Notably, there is an initial phase of random exploration (indicated by the vertical dashed red line) extending up to the first 250 observations. During this exploratory phase, the agent gathers data about the network state, which is used to populate the experience replay buffer (as outlined in Subsection 2.3.2.1 of Chapter 2).

After this initial exploration phase, the agent commences taking actions based on its learned knowledge, drawing from the experiences stored in the experience replay buffer. It's important to highlight that this buffer is continually updated, enabling the agent to learn from new states. Consequently, the agent can adapt to previously unseen states, a capability that proves particularly valuable in scenarios with non-stationary loads.

This analysis of the agent's actions provides insights into its learning process and the transition from exploration to exploitation as it becomes more knowledgeable about the environment.

Analyzing the agent's actions, it becomes apparent that during the initial phase of the experiment, characterized by an increase in network load, the agent frequently opted to increase the value of the target delay. Subsequently, as the load stabilized, the agent chose to take no action, potentially reducing the overhead of control plane operations in the data plane. Towards the end of the experiment, as the network load decreased, the agent shifted its strategy towards reducing the target delay.

Having observed how these actions mirror the agent's interactions with the environment, we can now delve deeper into the model's performance. Figure 31 provides an overview of the model's behavior, illustrated by the curves representing key performance metrics such as Loss and Reward.



(a) Loss (b) Reward

Figure 31 – Model performance results - Decreasing Loss and increasing Reward indicate model convergence.

Figure 31(a) illustrates the trajectory of Loss throughout the experiment. A decline in Loss signifies a lower Mean Squared Error (MSE) in predicting q-values. In essence, a low Loss value suggests that the model is effectively learning the policy by selecting actions that maximize rewards (QoS). During the initial phase of filling the experience replay buffer, Loss tends to be higher as actions are taken without the benefit of learning,

effectively representing random actions. However, as the experience replay buffer becomes populated and the Q-network is updated based on these experiences, the agent begins to make more informed and assertive decisions. This shift towards lower Loss values reflects the agent's ability to learn and improve its policy.

Turning our attention to Rewards (Figure 31(b)), we observe that the model incurs some penalties during the initial phase of the experiment. This corresponds to the period when the agent transitions from an initial stationary state with no charges to reaching the peak of the sinusoidal load curve, marked by the presence of 40 instances of the video player simultaneously. Subsequently, as the agent refines its decision-making, it starts receiving rewards consistently. These rewards indicate that the agent effectively maximizes the QoS of DASH, further underscoring the model's learning and adaptive capabilities.

The insights obtained from the agent's performance analysis are supported by the LBO and FPS metrics observed by the video client in response to DESiRED's actions, as outlined in Table 8 and depicted in Figure 32. At this conjuncture, we aim to provide an interpretation of the results from the video client's perspective, highlighting how DESiRED outperformed other approaches considered in this study.

An essential piece of data when evaluating the QoS of a video service is the resolution displayed on the screen by the video player. In this context, video consumers were offered three distinct quality levels:

1. Minimum Resolution: 426x240 pixels at 18 FPS.
2. Medium Resolution: 854x480 pixels at 24 FPS.
3. Maximum Resolution: 1280x720 pixels at 30 FPS.

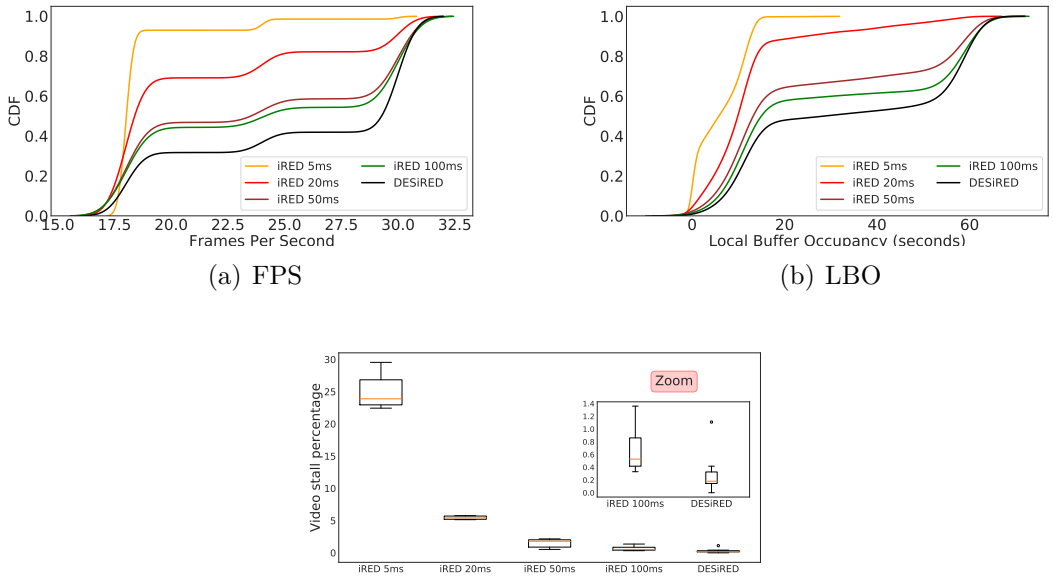
AQM	Min. Resolution	Med. Resolution	Max. Resolution
iRED 5ms	91.83%	5.49%	1.36%
iRED 20ms	68.77%	12.96%	17.69%
iRED 50ms	46.74%	11.73%	41.27%
iRED 100ms	43.81%	9.91%	46.18%
DESiRED	31.71%	10.15%	58.07%

Table 8 – Execution percentage at each video quality level.

Even under challenging conditions, Table 8 demonstrates that DESiRED exhibits the highest percentage of video playback at the maximum resolution (58.07%) and the lowest rate of playback at the minimum resolution (31.71%). This finding aligns with the data presented in Figure 32.

The discussion initiated in Subsection 6.3.1.2 remains pertinent in this context as well. To reiterate, during periods of intense competition for shared resources, probabilistic drops facilitated by a target delay that adjusts in response to network load fluctuations have

proven instrumental in maximizing the QoS of the video service. Once again, DESiRED effectively maintains a higher level of LBO filling, as depicted in Figure 32(b), ultimately contributing to superior FPS performance, as evidenced in Figure 32(a).



(a) FPS

(b) LBO

(c) Video Stall

Figure 32 – QoS measurements of the MPEG-DASH video service. DESiRED improves FPS and LBO while minimizing video stall.

Figure 32(c) presents a boxplot representing the percentage of video stalls, which signifies moments when the video remains frozen without any frames being displayed. A fast analysis of this figure might lead to the incorrect assumption that a longer delay at a fixed target would yield better results. However, it's important to note that DESiRED imposes an upper limit of 70ms, which is lower than the value employed by iRED100ms, thereby dispelling this theory. In this context, we believe that DESiRED's fine-tuned approach enables it to determine the optimal target delay value for each network state during the sinusoidal load.

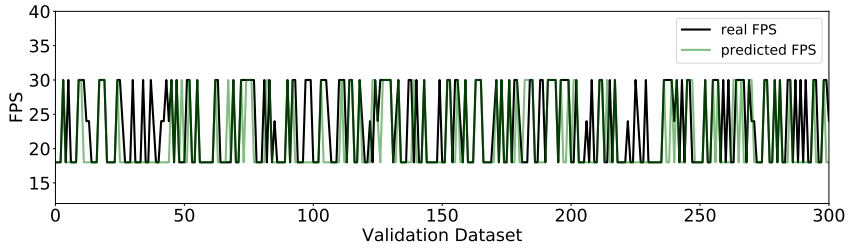
6.3.2 Case 2 - Using the data from the predicted video player

In this section, our goal is to evaluate the performance of DESiRED under conditions where the network service provider lacks access to the video player metrics. In such cases, we leverage supervised learning to predict the DASH QoS values. Based on our previous work (Almeida; Pasquini; Verdi, 2021) we chose the Random Forest Regressor as our estimator.

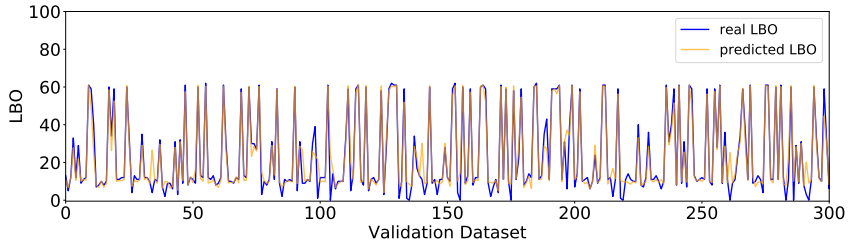
Initially, we use data (INT and QoS measurements) from execution under non-stationary load (Subsection 6.3.1.2) to train a regressor. About this topic, we undergo data pre-processing with the ultimate goal of finding the best regressor. We employed a random

search (Randomized search) with cross-validation (K-fold cross-validation) to identify the best parameters for a regressor in this dataset (Sinusoidal load).

After the pre-processing stage, we compute the NMAE to evaluate the performance of the estimator in the validation[4] dataset. The results indicate an estimation error of **5% for FPS** and **9% for LBO**. In Figure 33, we can observe a visual arrangement of the predictions made by the estimator in the validation set for FPS (Fig. 33(a)) and LBO (Fig. 33(b)).



(a) FPS



(b) LBO

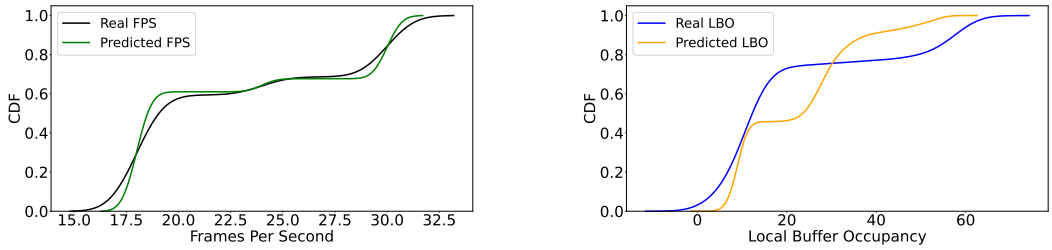
Figure 33 – QoS estimations of the trained regressor.

We attribute the smaller prediction error about FPS to the fact that our experiment only involved three levels of video quality (18, 24, and 30 FPS). On the other hand, with LBO varying between 0 and approximately 60 seconds, the regressor tends to make more errors in its predictions, thus justifying this difference in the NMAE value.

After this initial analysis of the performance of the regressor on the validation set, we evaluated DESiRED using the QoS values (FPS and LBO) predicted by this regressor as input. In other words, at this point, DESiRED calculates the rewards defined in its policy based on these estimates.

For an hour, we conduct DASH transmission and expose the network to the same non-stationary load pattern. We collect the real FPS and LBO values computed in the video player and compare them to the values estimated by our regressor. Fig. 34 summarizes the results of this experiment, in which we observed a similarity in the behavior of the CDF curve between the real and predicted FPS values (Fig. 34(a)), which can be reinforced by the NMAE of just 5%. Regarding the behavior of the CDF curve for LBO, we observed that the regressor has difficulty estimating values with greater precision during moments

[4] Samples not yet seen by the regressor in the training phase.



(a) FPS (b) LBO

Figure 34 – QoS estimations of trained regressor and real QoS values.

of low network load (high LBO). We believe this difficulty arises due to the low occupancy of buffers in the network, often approaching zero, justifying a higher NMAE (9%).

These results indicate the feasibility of using DESiRED in environments where the network service provider lacks access to the video client's performance metrics. The idea behind this solution is to train a model to predict FPS and LBO so that this pre-trained agent can guide the DRL to make decisions

6.4 Summary of the Chapter

This chapter introduced DESiRED, a solution to tackle the fixed target delay problem. By leveraging advanced network telemetry and the capabilities of deep reinforcement learning, DESiRED emerges as a solution to improve the DASH QoS. In this work, DESiRED represents the materialization of the Smart Closed Loop, encompassing all components within the **MAPE** cycle.

Chapter 7

Final Considerations

In this chapter, we present the conclusions of this thesis, describing the contributions, weaknesses, and future directions. Initially, we will revisit the hypothesis defined in Chapter 1 and discuss how we concluded that it is correct. We will also present the weaknesses of our solution, pointing out cases where it might not perform adequately. Moreover, we outline some directions for open issues not yet solved in this work.

7.1 Conclusions

The main conclusion of this thesis indicates that the hypothesis outlined at the beginning of this work is correct, that is, fine-grained measurements provided by In-band Network Telemetry (INT) can be successfully used as input for Machine Learning (ML) models to guide the Smart Closed Loop in a DASH transmission scope. This conclusion results from the progress made in the estimation of adaptive video service metrics, followed by the probabilistic packet discarding approach implemented with iRED, and culminating in the materialization of the Smart Closed Loop with DESiRED. The Smart Closed Loop concept was conceived within the MAPE (Monitor-Analyze-Plan-Execute) cycle, wherein components interact with the ultimate goal of continuous monitoring and management. We believe that this control loop could prove beneficial within ISP networks wherein the network provider maintains management over the interconnect devices. Additionally, it may find utility in scenarios where the video content provider also possesses ownership of the network infrastructure (Lam, 2021).

Seeking to evaluate the hypothesis, we listed three research questions, which were answered throughout the chapters of this work. The initial research question pertained to the difficulties associated with estimating metrics from an adaptive video service within

the context of a programmable network. This estimation relied on the utilization of fine-grained metrics supplied by INT as input for training regressors based on supervised learning.

The subsequent research question examined the feasibility of implementing real-time network management actions aimed at enhancing the Quality of Service (QoS) for a DASH service. We tackle this question through the development and deployment of the ingress Random Early Detection (iRED) approach. This approach represents a disaggregated P4-AQM mechanism designed to align with the L4S framework.

The final research question delved into the possibility of designing a comprehensive solution that integrates both control and data planes, thereby realizing a closed-loop mechanism. As a fundamental requirement, the control plane is envisioned to have some intelligence, leveraging machine learning algorithms to guide its operations. In the context of this topic, we introduced DESiRED as a viable Smart Closed Loop solution. DESiRED was founded on the composition of In-band Network Telemetry (INT) measurements, serving as input to Deep Reinforcement Learning (DRL) models. This approach aids in the decision-making process for optimizing a dynamic Active Queue Management (AQM) mechanism within a programmable network.

7.2 Weaknesses and Limitations

Despite the advances presented in this work, this section shows a set of limitations and weaknesses that still demand greater attention. First of all, our solution has not been evaluated with applications other than DASH. These services may be adversely affected by the probabilistic dropping, especially throughput-sensitive applications. This weakness opens up a new and challenging field of research since a plurality of applications and services with different requirements must coexist in a network.

Another point of attention arises for more in-depth evaluations concerning a greater variety of network conditions. Given the vast diversity of services, applications, topological configurations, and network loads encountered, it is imperative to acknowledge that an agent trained within a specific network environment cannot be expected to replicate its performance in other heterogeneous settings. In response to this challenge, Transfer Learning (TL) has emerged as a promising approach, aiming to address several intricacies not typically encountered in the context of RL.

However, the application of TL within an RL framework is a non-trivial undertaking, necessitating numerous adaptations to enable the agent to effectively leverage knowledge acquired in a source domain for application in a target domain. In this context, numerous questions arise, including but not limited to:

- ❏ What types of knowledge are amenable to successful transfer?

- ❏ Which RL structures are best suited for integration into a TL framework?
- ❏ What distinguishes a source domain from a target domain?

The literature (Zhu; Lin; Zhou, 2020) has briefly discussed this topic; however, we understand that a dedicated examination of these issues is needed within the specific context of transfer learning in RL, particularly in computer networking problem domains.

7.3 Future Directions and Open Issues

Despite the advancements outlined in this thesis, some still require further analysis, thus characterizing them as prospective challenges for the future. The first among these challenges pertains to the design and implementation of a Smart Closed Loop within the data plane. We have noted some contributions to deploying machine learning (ML) models exclusively in the data plane, such as IIsy (Zheng et al., 2022a), Planter (Zheng et al., 2022b) and QCMP (Zheng; Rienecker; Zilberman, 2023). However, these studies concentrate solely on conducting the inference phase within the data plane, relegating model training to the control plane. This is primarily due to the ample resources and mathematical functions that are available in the control plane. Moreover, there is still progress to be made before more intricate models like Deep Reinforcement Learning (DRL) can be effectively employed.

The second issue revolves around the generalization capacity of the Smart Closed Loop. While DESiRED was conceived to be application-agnostic, it is a challenge to adapt its reward policies to different types of applications. A more advanced approach might involve crafting a policy with the capability to dynamically adjust to the specific services being monitored by DESiRED.

Another topic that can be better explored in future studies is the use of Generative Adversarial Networks (GANs) to assist in the training, hyperparameterization, and fine-tuning of machine learning models. Recent preliminary studies (Tavares et al., 2024) demonstrate that, in some specific cases, generative AI can help with the operation and management of networks with machine learning models.

The use of multiple queues was not addressed in this work. However, it is naive to think that these cases do not exist. In this sense, it is necessary to investigate more deeply how the strategies discussed here would behave in cases where traffic visits multiple queues on the same node.

Bibliography

Alizadeh, M. et al. Data center tcp (dctcp). In: **Proceedings of the ACM SIGCOMM 2010 Conference**. New York, NY, USA: Association for Computing Machinery, 2010. (SIGCOMM '10), p. 63–74. ISBN 9781450302012. Available on: <https://doi.org/10.1145/1851182.1851192>.

Almeida, L. et al. Wave - um gerador de cargas múltiplas para experimentação em redes de computadores. In: **Anais Estendidos do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. Porto Alegre, RS, Brasil: SBC, 2023. p. 9–16. ISSN 2177-9384.

Almeida, L.; Verdi, F.; Pasquini, R. Estimando métricas de serviço através de in-band network telemetry. In: **Anais do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. Porto Alegre, RS, Brasil: SBC, 2021. p. 252–265. ISSN 2177-9384. Available on: <https://sol.sbc.org.br/index.php/sbrc/article/view/16725>.

Almeida, L. C. de et al. ired: Improving the dash qos by dropping packets in programmable data planes. In: **2022 18th International Conference on Network and Service Management (CNSM)**. 2022. p. 136–144.

_____. **iRED: A disaggregated P4-AQM fully implemented in programmable data plane hardware**. 2023.

Almeida, L. C. de; Pasquini, R.; Verdi, F. L. Using machine learning and in-band network telemetry for service metrics estimation. In: **2021 IEEE 10th International Conference on Cloud Networking (CloudNet)**. 2021. p. 33–39.

Ari, I. et al. Managing flash crowds on the internet. In: **MASCOTS 2003**. 2003. p. 246– 249. ISBN 0-7695-2039-1.

Arslan, S.; McKeown, N. Switches know the exact amount of congestion. In: **Proceedings of the 2019 Workshop on Buffer Sizing**. New York, NY, USA: Association for Computing Machinery, 2019. (BS '19). ISBN 9781450377454. Available on: <https://doi.org/10.1145/3375235.3375245>.

Bentaleb, A. et al. A survey on bitrate adaptation schemes for streaming media over http. **IEEE Communications Surveys Tutorials**, v. 21, n. 1, p. 562–585, 2019.

Bosshart, P. et al. P4: Programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, jul 2014. ISSN 0146-4833. Available on: <https://doi.org/10.1145/2656877.2656890>.

_____. P4: Programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, jul. 2014. ISSN 0146-4833. Available on: <https://doi.org/10.1145/2656877.2656890>.

Boutaba, R. et al. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. **Journal of Internet Services and Applications**, Springer, v. 9, n. 1, p. 1–99, 2018.

Briscoe, B. et al. Implementing the ' prague requirements ' for low latency low loss scalable throughput (l 4 s). In: **Netdev 0x13, THE Technical Conference on Linux Networking**. 2018. Available on: <https://api.semanticscholar.org/CorpusID:180259180>.

_____. **RFC 9330: Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture**. USA: RFC Editor, 2023.

Broadcom. **NPL - Network Programming Language Specification**. 2019.

_____. **Trident4 / BCM56880 Series**. 2023. Available on: <https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56880-series>.

Busse-Grawitz, C. et al. **pForest: In-Network Inference with Random Forests**. 2022.

Calasans, M. **DASH sobre OpenFlow: estimando métricas de QoS a partir da rede**. 83 p. Tese (Doutorado) — Universidade Federal de Uberlândia, Uberlândia - MG - Brazil, 2020.

Cardwell, N. et al. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. **Queue**, Association for Computing Machinery, New York, NY, USA, v. 14, n. 5, p. 20–53, out. 2016. ISSN 1542-7730. Available on: <https://doi.org/10.1145/3012426.3022184>.

Casado, M. et al. Ethane: Taking control of the enterprise. In: **Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications**. New York, NY, USA: Association for Computing Machinery, 2007. (SIGCOMM '07), p. 1–12. ISBN 9781595937131. Available on: <https://doi.org/10.1145/1282380.1282382>.

Castro, A. Góes de et al. Orchestrating in-band data plane telemetry with machine learning. **IEEE Communications Letters**, v. 23, p. 20, 10 2019.

Chen, X. et al. Fine-grained queue measurement in the data plane. In: **Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies**. New York, NY, USA: Association for Computing Machinery, 2019. (CoNEXT '19), p. 15–29. ISBN 9781450369985. Available on: <https://doi.org/10.1145/3359989.3365408>.

Chen, Y.; Wu, K.; Zhang, Q. From qos to qoe: A tutorial on video quality assessment. **IEEE Communications Surveys Tutorials**, v. 17, n. 2, p. 1126–1165, 2015.

Chiesa, M. et al. Purr: A primitive for reconfigurable fast reroute: Hope for the best and program for the worst. In: **Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies**. New York, NY, USA: Association for Computing Machinery, 2019. (CoNEXT '19), p. 1–14. ISBN 9781450369985. Available on: <https://doi.org/10.1145/3359989.3365410>.

Cisco. **Internet Report**. 2021. Available on: <https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/index.html>.

Doan, T. V. et al. Interplay between priority queues and controlled delay in programmable data planes. In: **2023 18th Wireless On-Demand Network Systems and Services Conference (WONS)**. 2023. p. 64–71.

Faceli, K. et al. **Artificial Intelligence: A Machine Learning Approach**. : LTC, 2011.

_____. **Inteligência artificial: uma abordagem de aprendizado de máquina**. : LTC, 2011.

Feng, W.-c. et al. The blue active queue management algorithms. **IEEE/ACM Trans. Netw.**, IEEE Press, v. 10, n. 4, p. 513–528, aug 2002. ISSN 1063-6692. Available on: <https://doi.org/10.1109/TNET.2002.801399>.

Floyd, S.; Jacobson, V. Random early detection gateways for congestion avoidance. **IEEE/ACM Transactions on Networking**, v. 1, n. 4, p. 397–413, 1993.

Garcia, L. F. U. et al. Introdução à linguagem p4 - teoria e prática. **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC) - Minicursos**, 2018. Available on: <http://143.54.25.88/index.php/sbrcminicursos/article/view/1769>.

Gettys, J. Bufferbloat: Dark buffers in the internet. **IEEE Internet Computing**, v. 15, n. 3, p. 96–96, 2011.

Gombos, G. et al. Active queue management on the tofino programmable switch: The (dual)pi2 case. In: **ICC 2022 - IEEE International Conference on Communications**. 2022. p. 1685–1691.

Goodfellow, I.; Bengio, Y.; Courville, A. Optimization for Training Deep Models. In: _____. **Deep Learning**. MIT Press, 2016. Chap. 8, p. 271–325. Available on: <http://www.deeplearningbook.org>.

Hafez, N. A.; Hassan, M. S.; Landolsi, T. Reinforcement learning-based rate adaptation in dynamic video streaming. **Telecommunication Systems**, v. 83, n. 4, p. 395–407, Aug 2023. ISSN 1572-9451. Available on: <https://doi.org/10.1007/s11235-023-01031-3>.

Hauser, F. et al. A survey on data plane programming with p4: Fundamentals, advances, and applied research. **Journal of Network and Computer Applications**, v. 212, p. 103561, 2023. ISSN 1084-8045. Available on: <https://www.sciencedirect.com/science/article/pii/S1084804522002028>.

Høiland-Jørgensen, T.; Täht, D.; Morton, J. Piece of cake: A comprehensive queue management solution for home gateways. In: **2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)**. 2018. p. 37–42.

Intel. **P4 16 Intel ® Tofino™ Native Architecture – Public Version**. 2021. Available on: <https://github.com/barefootnetworks/Open-Tofino/blob/master/PUBLIC_Tofino-Native-Arch-Document.pdf>.

ISO. Dynamic adaptive streaming over http (dash)-part 1: Media presentation description and segment formats. **ISO/IEC**, p. 23009–1, 2014.

Jacobson, V. Congestion avoidance and control. In: **Symposium Proceedings on Communications Architectures and Protocols**. New York, NY, USA: Association for Computing Machinery, 1988. (SIGCOMM '88), p. 314–329. ISBN 0897912799. Available on: <https://doi.org/10.1145/52324.52356>.

James, G. et al. **An Introduction to Statistical Learning**. : Springer, New York, NY, 2013. (Springer Texts in Statistics). ISBN 978-1-4614-7138-7.

Joshi, R. et al. Burstradar: Practical real-time microburst monitoring for datacenter networks. In: **Proceedings of the 9th Asia-Pacific Workshop on Systems**. New York, NY, USA: Association for Computing Machinery, 2018. (APSys '18). ISBN 9781450360067. Available on: <https://doi.org/10.1145/3265723.3265731>.

Kim, M.; Chung, K. Reinforcement Learning-Based adaptive streaming scheme with edge computing assistance. **Sensors (Basel)**, Switzerland, v. 22, n. 6, mar. 2022.

Kim, Y. et al. Buffer management of virtualized network slices for quality-of-service satisfaction. In: **2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. Verona, Italy: IEEE, 2018. (1, 18725013), p. 1–4.

Kotsiantis, S.; Kanellopoulos, D.; Pintelas, P. Data preprocessing for supervised learning. **International Journal of Computer Science**, v. 1, p. 111–117, 01 2006.

Kundel, R. et al. P4-codel: Active queue management in programmable data planes. In: **2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. 2018. p. 1–4.

_____. **P4-CoDel: Experiences on Programmable Data Plane Hardware**. 2021.

Lam, C. F. (invited) google fiber deployments: Lessons learned and future directions. In: **Optical Fiber Communication Conference (OFC) 2021**. Optica Publishing Group, 2021. p. Th4H.2. Available on: <https://opg.optica.org/abstract.cfm?URI=OFC-2021-Th4H.2>.

Lederer, S. **Why YouTube & Netflix use MPEG-DASH in HTML5**. 2015. <https://bitmovin.com/status-mpeg-dash-today-youtube-netflix-use-html5-beyond/>. Accessed: 2020-03-24.

Li, W. et al. Qtcp: Adaptive congestion control with reinforcement learning. **IEEE Transactions on Network Science and Engineering**, v. 6, n. 3, p. 445–458, 2019.

Lin, H. et al. Knn-q learning algorithm of bitrate adaptation for video streaming over http. In: **2020 Information Communication Technologies Conference (ICTC)**. 2020. p. 302–306.

Malangadan, N.; Raina, G.; Ghosh, D. **Synchronisation in TCP networks with Drop-Tail Queues**. 2023.

Marques, J. A.; Levchenko, K.; Gaspary, L. P. Responding to network failures at data-plane speeds with network programmability. In: **NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium**. 2023. p. 1–10.

McKeown, N. et al. Openflow: Enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Available on: <https://doi.org/10.1145/1355734.1355746>.

Menth, M. et al. Implementation and evaluation of activity-based congestion management using p4 (p4-abc). **Future Internet**, v. 11, n. 7, 2019. ISSN 1999-5903. Available on: <https://www.mdpi.com/1999-5903/11/7/159>.

Mnih, V. et al. Human-level control through deep reinforcement learning. **Nature**, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

Nembrini, S.; König, I. R.; Wright, M. N. The revival of the Gini importance? **Bioinformatics**, v. 34, n. 21, p. 3711–3718, 05 2018. ISSN 1367-4803. Available on: <https://doi.org/10.1093/bioinformatics/bty373>.

Nichols, K.; Jacobson, V. Controlling queue delay: A modern aqm is just one piece of the solution to bufferbloat. **Queue**, Association for Computing Machinery, New York, NY, USA, v. 10, n. 5, p. 20–34, may 2012. ISSN 1542-7730. Available on: <https://doi.org/10.1145/2208917.2209336>.

P4. **In-band Network Telemetry (INT) Dataplane Specification**. 2021.

Pan, R.; Natarajan, P.; Baker, F. **PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem**. 2015. Work in Progress. Available on: <https://datatracker.ietf.org/doc/draft-ietf-aqm-pie/03/>.

Papagianni, C.; Schepper, K. D. Pi2 for p4: An active queue management scheme for programmable data planes. In: **Proceedings of the 15th International Conference on Emerging Networking EXperiments and Technologies**. New York, NY, USA: Association for Computing Machinery, 2019. (CoNEXT '19 Companion), p. 84–86. ISBN 9781450370066. Available on: <https://doi.org/10.1145/3360468.3368189>.

Peterson, L.; Brakmo, L.; Davie, B. **TCP Congestion Control: A Systems Approach**. : Systems Approach, 2022.

Polyak, B. T.; Juditsky, A. B. Acceleration of stochastic approximation by averaging. **SIAM Journal on Control and Optimization**, v. 30, n. 4, p. 838–855, 1992. Available on: <https://doi.org/10.1137/0330046>.

Qiao, M.; Gao, D. Fine-grained active queue management in the data plane with p4. In: **2022 7th International Conference on Computer and Communication Systems (ICCCS)**. 2022. p. 174–179.

Rahouti, M. et al. A priority-based queueing mechanism in software-defined networking environments. In: **2021 IEEE 18th Annual Consumer Communications and Networking Conference (CCNC)**. 2021. p. 1–2.

Sandvine. **Sandvine Global Internet Phenomena Report**. 2023.

Schepper, K. D.; Briscoe, B.; White, G. **Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S)**. RFC Editor, 2023. RFC 9332. (Request for Comments, 9332). Available on: <https://www.rfc-editor.org/info/rfc9332>.

Spang, B. et al. Sammy: Smoothing video traffic to be a friendly internet neighbor. In: **Proceedings of the ACM SIGCOMM 2023 Conference**. New York, NY, USA: Association for Computing Machinery, 2023. (ACM SIGCOMM '23), p. 754–768. ISBN 9798400702365. Available on: <https://doi.org/10.1145/3603269.3604839>.

Spiteri, K.; Sitaraman, R.; Sparacio, D. From theory to practice: improving bitrate adaptation in the dash reference player. In: **ACM Transactions on Multimedia Computing, Communications, and ApplicationsVolume 15Issue 2s**. 2018. p. 123–137.

Spiteri, K.; Urgaonkar, R.; Sitaraman, R. K. Bola: Near-optimal bitrate adaptation for online videos. **IEEE/ACM Transactions on Networking**, v. 28, n. 4, p. 1698–1711, 2020.

Stadler, R.; Pasquini, R.; Fodor, V. Learning from network device statistics. **J. Netw. Syst. Manag.**, v. 25, n. 4, p. 672–698, 2017. Available on: <https://doi.org/10.1007/s10922-017-9426-z>.

Sutton, R. S.; Barto, A. G. **Reinforcement learning: An introduction**. : MIT press, 2018.

Tavares, T. C. et al. **TimeGAN as a Simulator for Reinforcement Learning Training in Programmable Data Planes**. 2024. Accepted into NOMS 2024.

Thomas, E. et al. Enhancing mpeg dash performance via server and network assistance. **SMPTE Motion Imaging Journal**, v. 126, n. 1, p. 22–27, 2017.

Toresson, L. **Making a Packet-value Based AQM on a Programmable Switch for Resource-sharing and Low Latency**. Tese (Doutorado) — Karlstad University, 2021. Available on: <https://urn.kb.se/resolve?urn=urn:nbn:se:kau:diva-82568>.

Verdi, F. L.; Luz, G. V. Infarr: In-network fast rerouting. **IEEE Transactions on Network and Service Management**, v. 20, n. 3, p. 2319–2330, 2023.

Watkins, C. J.; Dayan, P. Q-learning. **Machine learning**, Springer, v. 8, p. 279–292, 1992.

Wei, X. et al. Reinforcement learning-based qoe-oriented dynamic adaptive streaming framework. **Information Sciences**, v. 569, p. 786–803, 2021. ISSN 0020-0255. Available on: <https://www.sciencedirect.com/science/article/pii/S0020025521004588>.

Zheng, C.; Rienecker, B.; Zilberman, N. Qcmp: Load balancing via in-network reinforcement learning. In: **Proceedings of the 2nd ACM SIGCOMM Workshop on Future of Internet Routing & Addressing**. New York, NY, USA: Association for Computing Machinery, 2023. (FIRA '23), p. 35–40. ISBN 9798400702761. Available on: <https://doi.org/10.1145/3607504.3609291>.

Zheng, C. et al. **IIsy: Practical In-Network Classification**. 2022.

_____. **Automating In-Network Machine Learning**. 2022.

Zhu, Z.; Lin, K.; Zhou, J. Transfer learning in deep reinforcement learning: A survey. **CoRR**, abs/2009.07888, 2020. Available on: <https://arxiv.org/abs/2009.07888>.

Appendix

APPENDIX A

Achievements

A.1 Publications and disseminations during the Ph.D. (2019 - 2024)

- ALMEIDA, Leandro C. de; VERDI, Fábio L.; PASQUINI, Rafael. **Estimando métricas de serviço através de In-band Network Telemetry.** Brazilian Symposium on Computer Networks and Distributed Systems (SBRC 2021) Uberlândia, MG, Brazil.

- ALMEIDA, Leandro C. de; VERDI, Fábio L.; PASQUINI, Rafael. **Using Machine Learning and In-band Network Telemetry for Service Metrics Estimation**. 10th IEEE International Conference on Cloud Networking (IEEE CloudNet 2021), Virtual Conference.

- ALMEIDA, Leandro C. de; MATOS, Guilherme; PASQUINI, Rafael; PAPAGIANNI, Chrysa, VERDI, Fábio L. **iRED: Improving the DASH QoS by dropping packets in programmable data planes.** 18th International Conference on Network and Service Management (CNSM). November 2022, Thessaloniki, Greece.

- ALMEIDA, Leandro C. de; MATOS, Guilherme; PASQUINI, Rafael; PAPAGIANNI, Chrysa, VERDI, Fábio L. **Improving the DASH QoS by dropping packets in programmable data planes.** IEEE Global Communications Conference (IEEE Globecom), DEMO Session, Rio de Janeiro - Brazil, 2022.

- ALMEIDA, Leandro C. de; SILVA, Jefferson; LINS, Ricardo; JR, Paulo Ditarso Maciel; PASQUINI, Rafael; VERDI, Fábio L. **WAVE - Um gerador de cargas**

múltiplas para experimentação em redes de computadores. Brazilian Symposium on Computer Networks and Distributed Systems (SBRC 2023 -Tool Salon), Brasília - Brazil.

A.2 Co-authored publications and dissemination during the Ph.D. (2019 - 2024)

- SILVA, Lucas Carvalho; SILVA, Kadu Almeida; BATISTA, Leandro Henrique; MACIEL JR., Paulo Ditarso; ALMEIDA, Leandro Cavalcanti; SILVA, Thiago Gouveia. **Proposta de uma arquitetura extensiva ao protocolo BGP com balanceamento de carga através de múltiplos caminhos.** WORKSHOP DE PESQUISA EXPERIMENTAL DA INTERNET DO FUTURO (WPEIF - SBRC), 2020, Rio de Janeiro, RJ, Brazil.
- Matos, Guilherme; Almeida, Leandro; Contreras, Luis M.; Verdi, Fábio L. **INCA: a mechanism for traffic identification and chaining in the data plane**. 13th Latin-American Conference on Communications (LATINCOM 2021), Virtual Conference.
- Matos, Guilherme; Almeida, Leandro; Contreras, Luis M.; Verdi, Fábio L. **When SRv6 meets 5G Core: Implementation and Deployment of a Network Service Chaining Function in SmartNICs**. P4 Workshop, Virtual Conference, May 2021.
- LUZ, Gustavo; ROCHA, André L. B.; ALMEIDA, Leandro C. de; VERDI, Fábio L. **InFaRR: Um algoritmo para reroteamento rápido em planos de dados programáveis.** Brazilian Symposium on Computer Networks and Distributed Systems (SBRC 2022), Fortaleza, CE, Brazil.
- TAVARES, Thiago C.; ALMEIDA, Leandro C. de; SILVA, Washington Rodrigo Dias da; CHIESA, Marco; VERDI, Fábio L. **TimeGAN as a Simulator for Reinforcement Learning Training in Programmable Data Planes**. (IEEE Network Operations and Management Symposium 2024 (NOMS) - Accept)

A.3 Works under review

- ALMEIDA, Leandro C. de; SILVA, Washington Rodrigo Dias da; TAVARES, Thiago C.; PASQUINI, Rafael; PAPAGIANNI, Chrysa, VERDI, Fábio L. **DESiRED - Dynamic, Enhanced, and Smart iRED: A P4-AQM with Deep Reinforcement Learning and In-band Network Telemetry.** (Computer Networks - major review).

A.4 Other

- Almeida, Leandro, P. D. Maciel and F. Verdi. **Cloud Network Slicing: A systematic mapping study from scientific publications.** ArXiv, 2020.
- ALMEIDA, Leandro C. de; PASQUINI, Rafael; PAPAGIANNI, Chrysa, VERDI, Fábio L. **iRED: A disaggregated P4-AQM fully implemented in programmable data plane hardware.** Preprint 2023. (TNSM - under submission).
- DIMOGLIS, Angelos; PAPADOPOULOS, Konstatinos; ALMEIDA, Leandro C. de; PAPAGIANNI, Chrysa; PAPADIMITRIOU, Panagiotis; GROSSO, Paola. **Lightweight INT on the Tofino programmable switch.** 2024. (under submission).

APPENDIX B

Evaluation Description (Service Estimation)

We established a virtualized environment hosted on a Dell EMC PowerEdge R720 physical server model, featuring 2 Intel Xeon processors® E5-2630 v2 running at 2.60GHz, with 6 cores per socket (24 virtual CPUs), 48GB of RAM, a 2TB HDD, and operating on Ubuntu 18.04.5 LTS.

For virtualization, we utilized VirtualBox version 6.1.8 as the hypervisor, complemented by the use of Vagrant version 2.2.13 and Ansible version 2.9.15 for infrastructure provisioning. All associated artifacts and resources have been made accessible for replication purposes via a public repository[1].

B.0.1 Components description

The topology described in Table 9 and shown in Figure 35 is composed by 10 virtual machines having all their connections provided by BMv2 switches, which are P4-capable virtual equipments.

The dashServer is the component responsible for providing video streaming in the DASH standard for the client and the load generators. In this evaluation, two video streams were made available: the transmission of a soccer game for the client access; and a playlist containing the ten most accessed videos on Youtube® for the load generators. Apache version 2 applications were installed as the web server; FFmpeg (2.8.17) was used for encoding the videos; and MP4box (0.5.2) for creating the DASH manifest files.

[1] https://github.com/leandrocalmeida/

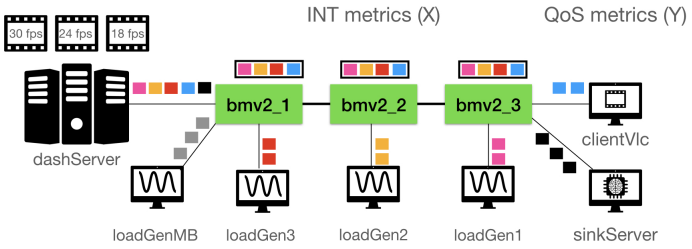


Figure 35 – Topology of experiment.

Name	OS	vCPUs	Memory
dashServer	Ubuntu 16.04.7 LTS	12	4GB
clientVlc	Ubuntu 20.04.1 LTS	12	8GB
sinkServer	Ubuntu 20.04.2 LTS	4	8GB
loadGen1	Ubuntu 20.04.1 LTS	6	8GB
loadGen2	Ubuntu 20.04.1 LTS	6	8GB
loadGen3	Ubuntu 20.04.1 LTS	6	8GB
loadGenMicroBurst	Ubuntu 20.04.1 LTS	1	4GB
bmv2_1	Ubuntu 20.04.1 LTS	4	1GB
bmv2_2	Ubuntu 20.04.1 LTS	4	1GB
bmv2_3	Ubuntu 20.04.1 LTS	4	1GB

Table 9 – VMs details.

The clientVlc is the component responsible for consuming the video streaming of the soccer game, in which the VLC video player (3.0.8) was executed, with modifications to collect service metrics.

The BMv2 switches were programmed to append INT metadata in all INT packets. In this experiment, we adopt an ONT approach, that is, specific INT probes are sent from the DASH server to the sink node. So, no data packets are changed to carry out INT metadata.

The sinkServer is the component responsible for collecting INT traffic and storing it in the format supported by the ML methods. Code written in Python was used to perform the collection and storage functionalities.

The load generators (loadGen{1/2/3}) are components responsible for consuming the streaming of the ten most accessed videos on Youtube®. They run two load patterns: a sinusoidal and flashcrowd.

The sinusoidal function is described in Equation 8, where: A represents an amplitude; F the frequency; and λ is a phase in radians. The loadGen{1/2/3} execute video clients obeying the sinusoid load function, increasing and decreasing over time.

$$f(y) = A\sin(F + \lambda) \tag{8}$$

The flashcrowd load describes a flash event, that is represented by a large spike or surge in traffic to a particular Web site (Ari et al., 2003). The flashcrowd is divided into

three phases: ramp-up, sustained and ramp-down.

Ramp-up is modeled by shock level (S), that is an order of magnitude increase in the average request (video clients) rate. Furthermore, it starts in t_0 and ends in t_1.

$$rampup = \frac{1}{\log_{10}(1+S)} \tag{9}$$

Sustained represents the maximum traffic (clients) level at the time interval t_1 and t_2. It is also modeled by S.

$$sustained = \log_{10}(1+S) \tag{10}$$

Ramp-down represents the end of the flash event, gradually decreasing the amount of traffic (video clients). In this phase, n is a constant that defines the speed of reduction. Ramp-down is modeled by n and S.

$$rampdown = n \times \log_{10}(1+S) \tag{11}$$

The loadGenMicroBurst is a component that runs a microburst (Joshi et al., 2018) generator used to create noise in the network load. The purpose of having microbursts is to mimic as much as possible the real traffic in a datacenter and make it difficult for the ML methods to characterize the load patterns.

B.0.2 Experiment description

The experiment lasted approximately 19 hours (8h for sinusoid, 6h for flashcrowd and 5h for mix load). Mix load means that both sinusoid and flashcrowd were used at the same time in the experiment. The dashServer hosts the video with different configurations (from high quality to low quality), as shown in Table 10, so that the client can use each one (transition) depending on the traffic load in the network.

Type	**Resolution**	**FPS**	**GOP**	**Kbps**	**Buffer**	**Codec**
vídeo	426x240	18	72	280	140	h264
vídeo	854x480	24	96	980	490	h264
vídeo	1280x720	30	120	2080	1040	h264
áudio	-	-	-	128	-	AAC
áudio	-	-	-	64	-	AAC

Table 10 – Video parameters used in a dashServer.

Every second, service quality metrics were collected in the clientVlc component. In parallel, packets with INT instructions were sent from the dashServer to the sinkServer in every µs interval. INT metadata is then appended by each switch (bmv2_{1,2,3}) in the path as shown in Figure 36. At each hop, the INT packet size increases **32 bytes** (metadata) from its original size (48 bytes).

Upon arriving at the sinkServer, the metadata is extracted and stored in the proper format for the ML methods. In addition, the load generators (loadGen{1,2,3}) run the

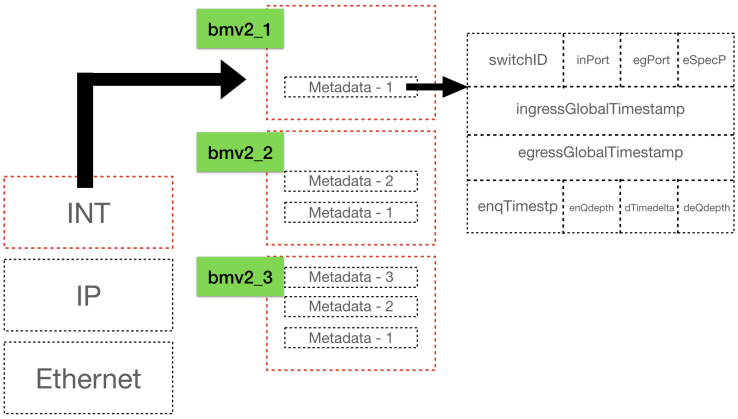


Figure 36 – Add INT metadata.

load patterns, sinusoid, flashcrowd[2] and mix, in independent executions. The parameters used are described in Table 11.

Component	Sinusoid	Flashcrowd	Mix
loadGen1	A=4, F=15, λ=5	S=(8-15), n=(1-5)	A=2, F=7, λ=5, S=(3-8), n=(1-2)
loadGen2	A=4, F=15, λ=5	S=(8-15), n=(1-5)	A=2, F=7, λ=5, S=(3-8), n=(1-2)
loadGen3	A=4, F=15, λ=5	S=(8-15), n=(1-5)	A=2, F=7, λ=5, S=(3-8), n=(1-2)

Table 11 – Parameters used for load generators.

The microburst generator sent 500 bytes-packets in a time interval between 0.01 - 1 second during all experiments.

After performing the experiments, the data (FPS and INT metrics) were integrated into a matrix M_{mxn}, where: m represents the number of samples (time series); and n represents the number of attributes used for the regression. Before submitting the data to the ML method, a pre-processing step was performed, in which the objective was to improve the quality and representation of the data. Pre-processing was carried out following the steps described below:

Incomplete/missing data handling: Incomplete/missing data (*NaN - Not a Number*) may lead to problems in the execution of the methods (Faceli et al., 2011a). For this reason, through the function `removeNaN`, samples with these values were removed.

Removal of same-valued attributes: single-valued attributes do not have information that helps distinguish objects, so they are considered irrelevant (Faceli et al., 2011a). For this reason, single-valued attributes have been removed with the `removeSameValuedAttr` function.

Attribute normalization: attributes that have very different scales can cause problems in machine learning methods (Faceli et al., 2011a). For this reason, the *z-score* (Kotsiantis; Kanellopoulos; Pintelas, 2006) attribute normalization process was performed

[2] The value was chosen randomly from a range described in Table 11.

with the `StandardScaler` function of the Scikit-learn package. In this case, the mean of each attribute was equal to zero and standard deviation equal to one.

After the pre-processing step, the dataset was divided into three partitions (train/test and evaluation) using the function `train_test_split` from python's Scikit-learn package. In this sense, 80% of data went to training/test and 20% to evaluation. Train/test partition (80% from original) was split again using the `KFold` cross-validation function, with k=5, creating 5 sub-partitions, 4 for train and 1 for test. Each partition was submitted to the ML models of the Scikit-learn package, performing a grid search through the `RandomizedSearchCV()` function, in order to find the regressor with the smallest error. For each method, 75 ML models were analyzed (25 for each load), totalizing 300 models evaluated. The best estimator of each ML method was used to evaluate the data evaluation partition (20% from original) for each load.

APPENDIX C

Evaluation Description (Resource consumption analysis in Tofino2)

Environment description. Our testbed consists of a P4 programmable switch (Edgecore DCS810 - Tofino2). The switch connects two Linux hosts, Sender and Receiver, having 25Gbps of link capacity, as shown in Fig. 17. Seeking to analyze the coexistence and fairness between different versions of TCP, each end-host sends TCP Cubic and Prague flows. We conducted our experiments over different network conditions shown in Table 7, varying bandwidth, RTT and Maximum Transmission Unit (MTU). The bandwidth is emulated by the P4-switch using the port shaping feature. The base RTT is emulated in the Receiver by the *tc netem* tool, delaying the ACKs of TCP flows. The MTU is emulated in the end-hosts (Sender and Receiver) by the *ifconfig* tool. The traffic is generated by the *iperf* tool.

Load description. The load applied to the experiment is composed of 4 phases of 120 seconds each. In each phase, new flows enter the system, that is, starting with less load and ending with a high load (bottleneck condition), as used in (Gombos et al., 2022). The number of Cubic and Prague flows are shown in Table 12.

Table 12 – Load parameters

Phase	Relative time	Cubic Flows	Prague Flows
1	0	1	1
2	120	2	2
3	240	10	10
4	360	25	25

AQMs settings. We use a base TARGET DELAY of 20ms for all AQMs. For iRED, we set the minimum and maximum thresholds for queue delay, configuring 20 (TARGET

delay) and 40 ms respectively, following the rule of thumb to set the maximum threshold as at least twice the minimum (Floyd; Jacobson, 1993). For PI2, we set the TARGET delay (20ms), INTERVAL (15ms), α (0.3125) and β (3.125), following the parameters used in (Gombos et al., 2022). In P4-CoDel, we set the TARGET delay (20ms) and INTERVAL (100ms), following the values used in (Kundel et al., 2021).

Ghost Thread. Tofino2 provides a new feature that enables the observation of the queue depth at the Ingress block per packet. From the flexibility that is brought by this new feature, we created a modified iRED version (iRED+G), that obtains the Egress port queue depth at the Ingres block, and then, makes the decision and the dropping both at the Ingres block. The key difference here is that we needed to adapt the iRED to use the queue depth rather than the queue delay.

Tables in grayscale. All tables used to present the results are colored in grayscale, in which the range of values is between light (best value) and dark (worst value).

Table 13 – Number of dropped packets

Conf	iRED	PI2	CoDel	iRED+G
I	35597	58417	35861	37561
II	11311	22735	17947	11575
III	9495	7546	36602	45725
IV	4103	1802	9086	13266
V	27826	38060	21282	29016
VI	7625	19246	20665	8077
VII	6141	4538	33136	28367
VIII	7296	2378	15301	15612
IX	18314	26663	33044	21841
X	5455	11852	10430	5973
XI	4684	2639	23975	17669
XII	12080	1510	29870	19744

Number of dropped packets. All evaluations were performed based on the number of dropped packets in each configuration, detailed in Table 13. The variation of the numbers refers to the drop probability (randomness) for each AQM.

C.0.0.1 Wasted Memory

In this subsection, we detail the results of wasted memory for each configuration evaluated in Table 14. In the case of Egress-based AQMs, the wasted memory is calculated by doing *2 * the size of the packet* (1500 bytes in the Ingress Buffer + 1500 bytes in the Traffic Manager). For iRED, the wasted memory is computed by the sum of the length of the dropped (1500 bytes) and notification (48 bytes) packets, resulting in 1500+48 = 1548 bytes. For the iRED+G, the wasted memory is only the Ingress buffer, which is 1500 bytes. We conjecture that there is some internal memory used by the Ghost mechanism to share queue depth information between the Traffic Manager and Ingress, but it's an internal feature that is not exposed to the programmer.

In general, as can be seen in Table 14, Egress-based AQMs need more memory to perform drops, given the same load. This happens because the AQM operations (decision

Table 14 – Wasted Memory (MB)

Conf	iRED	PI2	CoDel	iRED+G
I	55.09	175.24	107.58	56.34
II	17.5	68.2	53.84	17.36
III	14.69	22.78	109.8	68.58
IV	6.34	5.4	27.24	19.89
V	23.59	60.89	34.05	23.21
VI	6.46	30.79	33.06	6.46
VII	5.19	7.26	53.01	22.69
VIII	6.15	3.8	24.48	12.48
IX	8.19	21.33	26.43	8.73
X	2.44	9.48	8.34	2.38
XI	2.09	2.11	19.18	7.06
XII	5.4	1.2	23.89	7.89

and action) are combined in the Egress block. As packets dropped by iRED only cross the Ingress block, there is up to 10x less memory usage (Configuration VII).

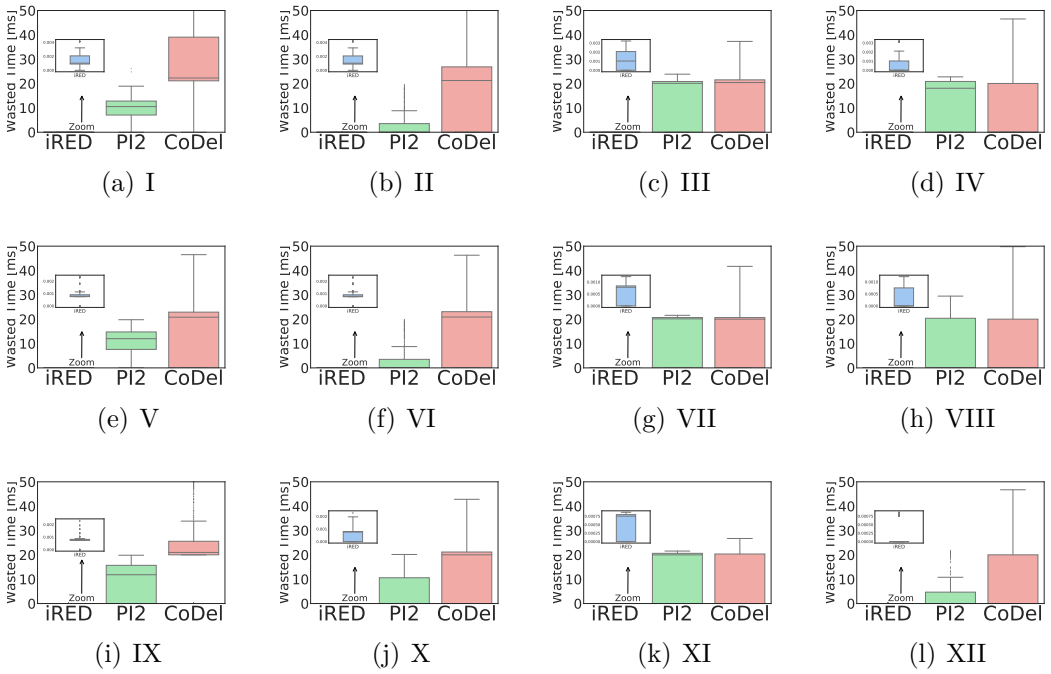
C.0.0.2 Wasted Time

In the case of the Egress-based AQM, the wasted time is defined by the queue delay computed for each discarded packet. In other words, it means the time that a given packet stayed in the output queue before being dropped. However, in TNA there is no intrinsic metadata to represent the queue delay. In this case, the traditional way (Kundel et al., 2021; Gombos et al., 2022) to do it is to compute the difference between *egress global timestamp (egTstmp)* and *ingress global timestamp (igTstmp)*. This difference represents the sum of the time spent in: Ingress parser latency; Ingress processing latency; Ingress deparser latency; and Traffic Manager latency. We create an internal bridge header to carry the *igTstmp* from Ingress to Egress, and when the packet reaches the Egress block, we get the *egTstmp* to calculate the queue delay.

In the Ingress-based AQMs, the discarded packets are not sent to the output queue, so **the queue delay is always zero**. However, the congestion notification needs to be carried to the Ingress block. iRED uses recirculation, so in this case, the wasted time is defined by the recirculation time for each notification packet sent from Egress to the Ingress block. Again, for the iRED+G, we were not able to compare it with the others, because it uses internal features that are not exposed to the programmer.

Fig. 37 shows the boxplot of the wasted time for iRED, P4-Codel and PI2. For reasons already explained, the iRED+G is not present in this measurement. In many of the observed cases for P4-CoDel and PI2, the median of the wasted time for the discarded packets is very close to the TARGET DELAY, that is, the packets waited in the queue for about 20ms before being discarded.

On the other hand, for iRED, the wasted time was very low, even requiring a zoom (blue boxplot) in the graph for better visualization of the measurements. In this case, only 48 bytes are transferred when the AQM logic decides to drop, that is, consumes very low time through the 400Gbps internal recirculation port.



(a) I (b) II (c) III (d) IV

(e) V (f) VI (g) VII (h) VIII

(i) IX (j) X (k) XI (l) XII

Figure 37 – Wasted Resources (Time).

Since iRED uses the high-speed recirculation port, the recirculation time is very small compared to the queuing delay of Egress-based AQMs. For example, the recirculation time was approximately 0.001ms per packet in all configurations evaluated, while dropped packets wasted 20ms on Egress-based AQMs. This explains why we need to zoom in on Fig. 37.

C.0.0.3 Wasted Latency and Power

The results shown in this section were obtained using the P4 Insight (*p4i*) tool[1] provided by Intel to inspect the P4-codes. First of all, by means of P4 Insigtht, we obtained the cycles and power consumption for each AQM. Table 15 summarizes the *p4i* output for each metric evaluated (for each programmable block).

The number of Cycles or Weight for iRED is more balanced between Ingress and Egress. Although for iRED the Cycles/Weight numbers are balanced between Ingress and Egress, the dropped packets essentially consumed the resources of the Ingress block. Not surprisingly, for PI2 and CoDel, most of the Cycles and Weights are concentrated in the Egress. Noteworthy to say that, although the number of cycles for PI2 is smaller than CoDel, the weight is larger for PI2. The explanation refers to the fact that PI2 needs additional registers to store the probabilities computed by the control plane, requiring more power consumption for the writing operations. For iRED+G, all AQM logic is concentrated in the Ingress.

[1] https://www.intel.com.br/content/www/br/pt/products/details/network-io/intelligent-fabric-processors/p4-insight.html.

Table 15 – Latency and Power

AQMs	Cycles		Weight	
	Ingress	Egress	Ingress	Egress
iRED	108.0	192.0	112.5	158.8
PI2	60.0	160.0	20.8	235.8
CoDel	60.0	196.0	13.8	154.9
iRED+G	212.0	84.0	208.0	31.6

Then, by having the numbers shown in Tab. 15, we were able to calculate the wasted cycles and weight.

C.0.0.4 Wasted Clock Cycles

Each block has a fixed number of clock cycles (Latency), which are necessary to forward each packet through the pipeline. For PI2, the wasted cycles are computed by $60 + 160 = 220$ cycles per dropped packet. For P4-CoDel, the wasted cycles are computed by $60 + 196 = 256$ cycles per dropped packet. In iRED and iRED+G cases, only Ingress cycles are used, resulting in 108 and 212 per dropped packet respectively.

Table 16 – Wasted Clock Cycles

Conf	iRED	PI2	CoDel	iRED+G
I	3844476.0	12851740.0	9180416.0	7962932.0
II	1221588.0	5001700.0	4594432.0	2453900.0
III	1025460.0	1660120.0	9370112.0	9693700.0
IV	443124.0	396440.0	2326016.0	2812392.0
V	3005208.0	8373200.0	5448192.0	6151392.0
VI	823500.0	4234120.0	5290240.0	1712324.0
VII	663228.0	998360.0	8482816.0	6013804.0
VIII	787968.0	523160.0	3917056.0	3309744.0
IX	1977912.0	5865860.0	8459264.0	4630292.0
X	589140.0	2607440.0	2670080.0	1266276.0
XI	505872.0	580580.0	6137600.0	3745828.0
XII	1304640.0	332200.0	7646720.0	4185728.0

In Table 16, the cycles consumed by iRED for the dropped packets are colored on a lighter scale in most parts of the configurations. If we look at the values, iRED achieves savings in the order of up to 12x fewer clock cycles. Moreover, the results of the iRED+G show that despite running in the Ingress, it wastes a large number of clock cycles for each dropped packet since all AQM logic operations are combined within the same programmable block.

C.0.0.5 Wasted Weight (Power Consumption)

The Wasted Weight is a sum of weights (Power consumption) in Ingress and Egress for each dropped packet. For PI2, the wasted weight is computed by $20.8 + 235.8 = 256,8$ per dropped packet. For P4-CoDel, the wasted weight is computed by $13.8 + 154.9 = 168,7$ per dropped packet. In iRED and iRED+G cases, only Ingress weights are used, resulting in 112.5 and 208 per dropped packet respectively.

Table 17 – Wasted Weight (Power Consumption)

Conf	iRED	PI2	CoDel	iRED+G
I	4004662.5	14989802.2	6049750.7	7812688.0
II	1272487.5	5674656.0	3027658.9	2407600.0
III	1068187.5	1883481.6	6174757.4	9510800.0
IV	461587.5	449779.2	1532808.2	2759328.0
V	3130425.0	9499776.0	3590273.4	6035328.0
VI	857812.5	4803801.6	3486185.5	1680016.0
VII	690862.5	1132684.8	5590043.2	5900336.0
VIII	820800.0	593548.8	2581278.7	3247296.0
IX	2060325.0	6655084.8	5574522.8	4542928.0
X	613687.5	2958259.2	1759541.0	1242384.0
XI	526950.0	658694.4	4044582.5	3675152.0
XII	1359000.0	376896.0	5039069.0	4106752.0

Looking at Table 17, the Egress-based AQMs have more power consumption in comparison to iRED, because all drop logic is not disaggregated. This is repeated with the iRED+G version, which concentrates all operations in the Ingress block. On the other hand, as iRED splits AQM's operations, only the Ingress block's power resources are consumed by dropped packets. Then, iRED reduces power consumption by up to 8x.

APPENDIX D

Evaluation Description (DESiRED)

In this appendix, we provide a comprehensive overview of the DESiRED evaluation. This encompasses a detailed exposition of the research methodology, an in-depth portrayal of the experimental environment and its configuration, the load pattern employed, the DRL mechanism implemented, the metrics and measurements used for comprehensive analysis.

D.0.1 Research methodology

Our methodology is rooted in experimental research aimed at evaluating the effectiveness of the DRL mechanism within DESiRED. In this experiment, our aim is to conduct a comprehensive evaluation of DESiRED in comparison to iRED, where iRED employs fixed target delay settings of 5ms, 20ms, 50ms, and 100ms. We evaluate these approaches under both stationary (low and high) and non-stationary (sinusoidal) load conditions. To mitigate potential biases, each round of the investigation, spanning one hour, was repeated ten times for each approach, resulting in a cumulative duration of over fifty hours across independent runs. Furthermore, to gauge DESiRED's robustness, we aggregated the DRL agents derived from all preceding executions by employing an ensemble approach. This involved combining the model parameters through an exponentially decaying running average, as described by Eq. 12 (Polyak; Juditsky, 1992; Goodfellow; Bengio; Courville, 2016):

$$\hat{\theta}^{(t)} = \alpha\hat{\theta}^{(t-1)} + (1-\alpha)\theta^{(t)} \tag{12}$$

where θ represents a parameter from the i-th Q-network at the gradient descent iteration t; $\hat{\theta}^{(t)}$ the average from all parameters ($\frac{1}{t}\Sigma_i\theta^{(i)}$); and α the exponential decaying factor

(defined as 2.0).

We evaluate the DASH performance from the client-side perspective, focusing on three key metrics: FPS, LBO, and Rebuffering Rate (Starvation) as measured within the video player. Higher values for FPS and LBO correspond to improved QoS, while for Rebuffering Rate, a lower value signifies enhanced QoS.

In addition to evaluating application quality metrics, we also scrutinize the performance metrics of the DRL agent, encompassing Loss function and Rewards.

D.0.2 Environment description

The experiment was constructed within a realistic testbed, adopting an Infrastructure as Code (IaC) approach, and implemented using Vagrant, Virtualbox (version 6.1.28), and Ansible (version 2.10.8). In this setup, each infrastructure component is represented by an isolated virtual machine, interlinked through a P4 programmable data plane network.

Each switch in the experiment was equipped with both the iRED and DESiRED approaches. On the control plane side, the DRL engine was implemented, comprising approximately 750 lines of code and utilizing Tensorflow as its backend framework. The CDN was deployed to facilitate an DASH service, featuring live streaming of a soccer game and a playlist housing the ten most frequently accessed YouTube videos. Load management was executed using WAVE (Almeida et al., 2023) [1], a versatile load generator that orchestrates instances of an application over time.

This infrastructure was hosted on a bare-metal server, namely the Dell EMC PowerEdge R720, equipped with 2 Intel Xeon processors (E5-2630 v2, 2.60GHz) boasting 6 cores per socket (amounting to 24 virtual CPUs), 48GB of RAM, a 2TB HDD, and running the Ubuntu 20.04.6 LTS operating system. All pertinent artifacts and resources can be accessed within the repository available at our GitHub[2].

The Server hosts video content using the DASH standard to both the Video Client and the Load Generator. It offers various configurations, as detailed in Table 18, with each configuration having a chunk segment size of 4 seconds. The Video Client dynamically selects and transitions between these configurations based on network traffic conditions and the adaptation logic embedded within the video player.

The infrastructure is equipped with Apache version 2 as the web server, FFmpeg (version 2.8.17) for video encoding, and MP4box (version 0.5.2) for creating the DASH manifest files, ensuring seamless video streaming.

The Video Client utilizes DASH.js, a contemporary DASH reference player equipped with an ABR algorithm. It employs this ABR algorithm to consume the video stream of the soccer game, with the TCP New Reno congestion control algorithm managing network congestion.

[1] https://github.com/ifpb/wave
[2] https://github.com/dcomp-leris/DESiRED.

Type	Resolution	FPS	Group of Pictures	Kbps	Buffer	Codec
vídeo	426x240	18	72	280	140	h264
vídeo	854x480	24	96	980	490	h264
vídeo	1280x720	30	120	2080	1040	h264
áudio	-	-	-	128	-	AAC
áudio	-	-	-	64	-	AAC

Table 18 – Video parameters used in an MPEG-DASH Server.

The Load Generator is responsible for introducing network noise, operating the WAVE framework with a variety of loads, including both stationary and non-stationary scenarios. It dynamically adjusts the number of video player instances over time to simulate changing network conditions. Further elaboration on this aspect can be found in Subsection D.0.3.

All the switches utilized in this experiment were implemented within the BMv2 software switch environment, incorporating the respective P4 code for both iRED (fixed target delay) and DESiRED (dynamic target delay with DRL) approaches. Across all approaches, telemetry instructions were meticulously programmed to append telemetry metadata to all probe packets. Notably, this experiment follows the ONT approach, wherein dedicated ONT probes are dispatched from the DASH server to the Video Client. Consequently, no modifications are made to data packets to accommodate telemetry metadata. The specifics of the telemetry metadata, consisting of 32 bytes, gathered at each node within this experiment, are elaborated upon in Table 19.

Name	bits	Description
Switch ID	31	the switch identification number
Ingress port	9	the port number that the packet entered in the switch
Egress port	9	the port number that the packet left of the switch
Egress spec	9	the port number (Ingress) in which the packet will leave the switch
Ingress Global Timestamp	48	the timestamp, in µs, of when the packet entered in the ingress
Egress Global Timestamp	48	the timestamp, in µs, of when the packet started processing in the egress
Enq Timestamp	32	the timestamp, in µs, of when the packet was enqueue
Enq Qdepth	19	the queue depth when the packet was queued
Deq Timedelta	32	the time, in µs, that the packet remained in the queue
Deq Qdepth	19	the queue depth when the packet was dequeued

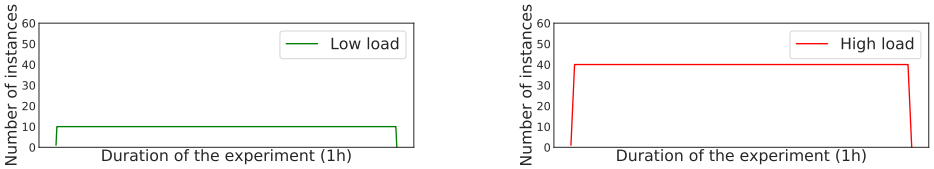
Table 19 – INT medatada.

D.0.3 Load Pattern

The Load Generator, powered by WAVE, orchestrates the instances of video clients over time based on input parameters described by a mathematical function that defines the load pattern. In its current iteration, WAVE supports constant, sinusoidal, and flashcrowd load patterns. It initiates and concludes video player processes, generating network load through genuine video requests (real traffic) that flow from the video player to the DASH Server.

In this study, our aim is to evaluate DESiRED under various load conditions, aiming to simulate diverse network state scenarios. To achieve this, we employ two distinct

categories of load patterns: stationary and non-stationary. For stationary loads, which remain constant throughout the experiment, we classify them into two types: low and high. In this context, a low load is characterized by the presence of ten video client instances operating concurrently throughout the duration of the experiment, as depicted in Figure 38(a). Conversely, a high load is characterized by the simultaneous operation of forty video player instances, representing a high-intensity load, as illustrated in Figure 38(b).



(a) Low Load (b) High Load

Figure 38 – Stationary Loads.

Under low load conditions, it is anticipated that the target delay will be attained relatively infrequently, given the shorter queuing delays that typically prevail. In this scenario, both AQM strategies, whether employing a fixed or dynamic target delay, are likely to yield comparable results in terms of QoS.

However, when the network experiences predominantly high load, the surge in traffic volume can lead to an increase in queue delay, thereby prompting AQM strategies to respond in accordance with the specified target delay, whether fixed or dynamic. In such instances, the dynamic adaptability of DESiRED's target delay is expected to confer advantages in terms of QoS compared to the rigid, fixed target delay approach employed by iRED. This dynamicity enables DESiRED to better accommodate and optimize QoS in the face of fluctuating and demanding network conditions.

It is indeed unrealistic to assume that network loads will always remain stationary or static. Consequently, in the second phase of our evaluation, we undertook a more comprehensive evaluation under a realistic load scenario, one that mirrors the dynamic nature of real-world network environments. Our objective was to evaluate non-stationary load patterns, encompassing both peak (high load) and off-peak (low load) periods within a single experiment. To achieve this, we employed a sinusoidal periodic load pattern characterized by the sinusoidal function detailed in Equation 13, where A represents the amplitude, F denotes the frequency, and λ signifies the phase in radians. The specific input parameters utilized for this evaluation were: $A = 15$, $F = 1$, and $\lambda = 25$, culminating in the load pattern illustrated in Figure 39. This approach captures the fluctuations in network load more realistically, offering a dynamic and challenging environment for our evaluation.

$$f(y) = A\sin(F + \lambda) \tag{13}$$

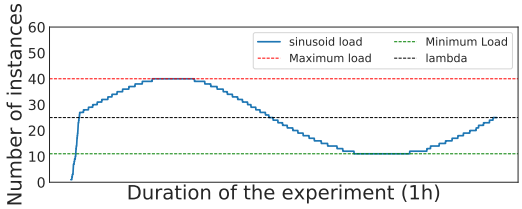


Figure 39 – Non-stationary (Sinusoid) Load.

D.0.4 Deep Reinforcement Learning mechanism

To accomplish the objectives outlined in this paper, we tailored the DQN architecture and agent-environment workflow to align with the distinctive characteristics of the DESiRED environment. In doing so, we designed the DQN using a Multilayer Perceptron (MLP) architecture, which is well-suited for handling the tabular nature of network telemetry metadata. The MLP network adopted in our approach consists of an input layer featuring units corresponding to each INT feature, two hidden layers each comprising 24 neurons, and an output layer containing units for each possible action that the agent can undertake, as depicted in Figure 40. Importantly, both the online and target networks share this identical architecture. Table 20 provides a detailed breakdown of the hyperparameters utilized for training DESiRED.

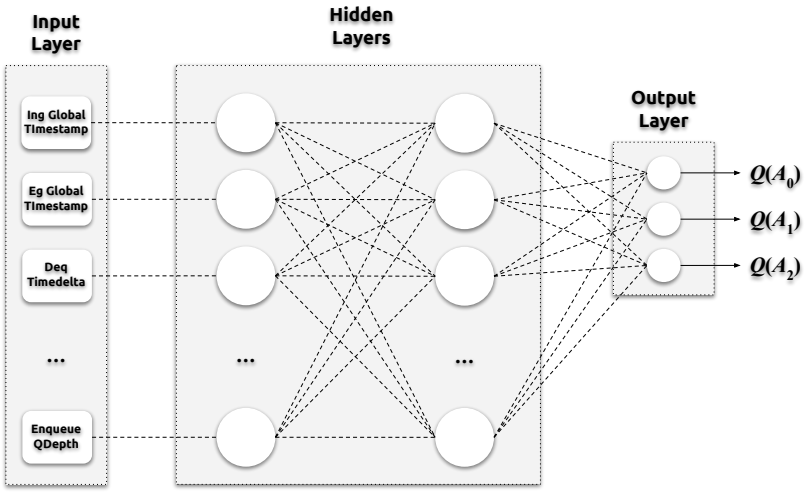


Figure 40 – DESiRED DQN architecture. The input layer is a network of fine-grained measurements, provided by INT. Hidden Layers make up the DQN. The actions are defined in the Output Layer.

To facilitate the desired agent-environment interaction, we formulated the agent's behavior as an MDP with the video chunk size serving as the discrete time steps. In this framework, DESiRED operates within the environment, dynamically adjusting the target delay in all switches at 4-second intervals, synchronized with the video chunk size. The agent's action space is delineated in Table 21, where it is evident that the action

Hyperparameter	Value	Description
Q-network input layer dimension	19	A scalar defining the state input shape.
Q-network hidden layers	2	A scalar defining the Q-network depth.
Q-network hidden units	24	A scalar defining the Q-network non-linear computing units.
Q-network output layer dimension	3	A scalar defining the Q-network predictions output shape.
Hidden units activation function	ReLU[1]	The non-linear activation function computed by hidden units.
Output units activation function	Linear	The activation function computed by the output layer.
Optimization function	SGD[2]	The function used to adjust the Q-network weights in order to minimize the predictions error in relation to the expected output.
SGD momentum	0.9	A scalar defining the momentum included in the optimization equation to accelerate the gradient descent.
Learning rate	1e-3	A scalar determining the pace at which the weights are updated.
Loss function	MSE[3]	The function used to compute the Bellman equation error.
Gamma	0.99	A scalar determining the discount factor in the Q-Learning update.
Tau	1e4	A scalar value determining how many updates the online network should perform before updating the target network (it corresponds to the parameter C depicted in the Fig. 10).
Experience replay capacity	1e6	A scalar defining the size of the list in which the agent's experience will be stored.
Experience replay minimum memory	100	A scalar defining the minimum experiences that should be stored before updating the online network.
Mini batch size	32	A scalar defining the number of experience samples over which the Q-network will be updated.
Starting epsilon	1.0	A scalar defining the initial probability to take random actions in the ϵ-greedy exploration.
Ending epsilon	0.01	A scalar defining the final probability to take random actions in the ϵ-greedy exploration.
Epsilon decay steps	250	A scalar determining how many steps the probability to take random actions in the ϵ-greedy exploration should decrease linearly before the exponential decay.
Epsilon exponential decay	0.99	A scalar determining exponential decay of the probability to take random actions in the ϵ-greedy exploration.

[1] Rectified Linear Unit.
[2] Stochastic Gradient Descent with Nesterov Momentum.
[3] Mean Squared Error.

Table 20 – DQN hyperparameters.

Action Number	Value	Description
0	+ 2ms	increase target delay in all switches until 70ms (upper limit)
1	- 1ms	decrease target delay in all switches until 20ms (lower limit)
2	-	do nothing

Table 21 – DESiRED actions space.

to increase the target delay brings about a modification that is proportionally twice as substantial as the decrease action. This choice was made to prompt DESiRED to respond promptly to transient congestion while retaining the flexibility to decrease the target delay when necessary, mirroring the rationale discussed in (Li et al., 2019).

It's important to highlight that the calculation of rewards does not occur immediately after an action is taken in the current state. This delay in reward calculation is attributed to the fact that the effects of the agent's action do not manifest instantly, primarily due to the inherent control mechanisms incorporated within TCP and ABR systems, as detailed in (Spiteri; Sitaraman; Sparacio, 2018). Consequently, the computation of rewards is deferred until the subsequent state's observation. In this context, the agent relies on network status data derived from INT measurements to form its states, selects actions, and is subsequently rewarded based on its ability to optimize the video's QoS, which is characterized by metrics such as FPS and LBO.

Indeed, the intrinsic correlation between metrics such as LBO and FPS presents a challenge when devising a reward policy. As the LBO increases, there is a tendency for the FPS to also increase. However, this relationship is not always straightforward due to the complex dynamics of network congestion and video streaming.

To calculate a reward (R_{t+1}) for a specific action (A_t), we adopt a strategy that first evaluates whether the LBO in the next state (LBO_{t+1}) improves compared to the LBO observed when the action was executed (LBO_t). Subsequently, a reward score is assigned based on the effects of this action on both the next state's LBO and FPS (FPS_{t+1}). Consequently, the agent receives maximum reward whenever the action taken leads to the maximization of LBO_{t+1} and is penalized in an inversely proportional manner if the video experiences stalls. The algorithmic logic for calculating rewards is detailed in Algorithm 3. This approach ensures that the agent's reward is contingent on its capacity to optimize both LBO and FPS, balancing the trade-offs inherent to video streaming in dynamic network conditions.

Algorithm 3 DESiRED reward policy algorithm.

```
if LBO_{t+1} > LBO_t then
    if LBO_{t+1} > 30 then
        R_{t+1} ← 2
    else if LBO_{t+1} < 30 then
        if FPS_{t+1} == 30 then
            R_{t+1} ← 1
        else if FPS_{t+1} == 24 then
            R_{t+1} ← 0.5
        else
            R_{t+1} ← 0.1
        end if
    end if
end if
if LBO_{t+1} < LBO_t then
    if LBO_{t+1} > 30 then
        R_{t+1} ← 2
    else if LBO_{t+1} < 30 then
        if FPS_{t+1} == 30 then
            R_{t+1} ← 1
        else if FPS_{t+1} == 24 then
            R_{t+1} ← 0.5
        else
            R_{t+1} ← −2
        end if
    end if
end if
```

These actions were executed according to the ϵ-greedy strategy as elucidated in Subsection 2.3.2.1. To implement this strategy, we established initial and final probabilities for taking random actions, specified the number of decaying steps, and defined an exponential decay factor (as outlined in Table 20). In this scheme, ϵ commences its linear decrease for 250 time steps to facilitate exploration. Subsequently, the probability of selecting random actions is exponentially reduced, gradually transitioning to a minimal value to emphasize exploitation over exploration. This strategy allows the agent to strike a balance between exploring new actions and exploiting its existing knowledge as it in-

teracts with the environment.

Taking into consideration the agent's action frequency of once every 4 seconds and the requirement for 250 iterations to initiate the exponential decay of ϵ, the exploration phase is expected to persist for approximately 17 minutes (equivalent to 1000 seconds). In tandem, the experience replay memory buffer necessitates a minimum of 100 samples to facilitate the online network parameter updates (as indicated in Table 20. Since experiences resulting from the agent-environment interaction are stored every 8 seconds, it would take approximately 13 minutes (or 800 seconds) for this condition to be met. Consequently, the online network undergoes an update each time a new experience is stored, as illustrated in Figures 9 and 10.

In the case of the non-stationary load, it follows a trajectory of 15 minutes to reach its peak, maintains a plateau for an additional 15 minutes, and subsequently begins to decline. During this period, the agent explores the action space during the ascending phase of the sinusoidal curve and exploits these actions during the plateau and descending phases. Consequently, when the exploitation stage commences, the agent should have already gleaned insights from past experiences, encompassing both low and high load scenarios. This enables the agent to adapt and respond effectively to the fluctuating network conditions.

D.0.5 Metrics and Measurements

On the video client side, we evaluate the QoS by monitoring key metrics, including:

- FPS (Frames Per Second): This metric quantifies the number of frames displayed per second on the screen, reflecting the smoothness of the video playback.
- LBO (Local Buffer Occupancy): LBO measures the remaining time, in seconds, for frames stored in the player's local buffer. It provides insights into the buffer's capacity to absorb network fluctuations and maintain continuous playback.

From these primary metrics, we derive additional insights, including:

- Resolution Distribution: We analyze the percentage of video content played at different resolutions (Maximum, Medium, and Minimum) to assess the adaptive streaming capabilities.
- Rebuffering Rate: This metric represents the percentage of time during which the video experiences stalls or freezes on the screen, indicating interruptions in playback.

To facilitate these measurements, we configure the DASH.js player to log these metrics on a per-second basis. Within the DRL mechanism, we focus on evaluating the performance metrics of the DQN:

- Loss: This metric is calculated as the MSE between the predicted q-values for the current and next states. It reflects the convergence and accuracy of the DQN's predictions.
- Reward: Reward represents the cumulative rewards and penalties acquired throughout the experiment. It offers insights into the agent's performance in maximizing QoS.

Additionally, we capture the action history for each experiment, documenting the agent's selected actions at each observation space (every 4 seconds). These metrics provide a comprehensive view of the agent's learning and adaptation throughout the experiment.