UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LEONARDO REINEHR GOBATTO

Improving Content-Aware Video Streaming in Congested Networks with In-Network Computing

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Engineering

Advisor: Prof. Dr. José Rodrigo Furlanetto Azambuja
Coadvisor: Prof. Dr. Weverton Luis da Costa Cordeiro

Porto Alegre
April 2023

“If I have seen farther than others,
it is because I stood on the shoulders of giants.”
— Sir Isaac Newton

ACKNOWLEDGEMENTS

I would like to thank my advisor José Azambuja and my co-advisor Weverton Cordeiro, who helped me with this and every other project. The meetings and conversations were essential to the project's success and to my personal career decisions.

I am grateful to all of those with whom I have had the pleasure to work during this and other related projects. Especially my friends Pablo Rodrigues, who introduced me to this research group, and Mateus Saquetti, who I have had the opportunity to work closely with these last years.

I would like to thank my parents Janete and Milton, and my sisters Luana and Michele, who have always supported and helped me. Thank you to several other relatives and friends that showed their support and whom I was also able to learn from.

Júlia, my partner through all these years, who always provides valuable feedback in all of my works, thank you for all the love and support. I love you!

ABSTRACT

Network congestion and packet loss pose an ever-increasing challenge to video streaming. Despite the research efforts toward making video encoding schemes resilient to lossy network conditions, forwarding devices have not considered monitoring packet content to prioritize packets and minimize the impact of packet loss on video transmission. In this work, we advocate in favor of in-network computing employing a packet drop algorithm and an in-network hardware module to devise a solution for improving content-aware video streaming in congested network. The results indicate a substantial decrease in frame loss (up to 7%), with minimal impact on resource utilization and performance costs.

Keywords: In-network Computing. networking. Video Streaming.

Melhorando a transmissão de vídeo em redes congestionadas utilizando In-Network Computing

RESUMO

O congestionamento da rede e a perda de pacotes representam um desafio cada vez maior para o streaming de vídeo. Apesar dos esforços de pesquisa para tornar os esquemas de codificação de vídeo resilientes a condições de rede com perdas, os dispositivos de encaminhamento não consideraram o monitoramento do conteúdo do pacote para priorizá-los e minimizar o impacto da perda de pacotes na transmissão de vídeo. Neste trabalho, defendemos a computação em rede empregando um algoritmo de descarte de pacotes e um módulo de hardware em rede para desenvolver uma solução para melhorar o streaming de vídeo com reconhecimento de conteúdo em uma rede congestionada. Os resultados obtidos indicam uma redução substancial na perda de quadros (até 7%), com mínimo impacto na utilização de recursos e performance.

Palavras-chave: In-network computing. Redes de Computadores. Transmissão de Vídeo.

LIST OF ABBREVIATIONS AND ACRONYMS

ASIC	Application Specific Integrated Circuit
AVC	Advanced Video Coding
CLB	Configurable Logic Blocks
CTC	Common Test Conditions
DSL	Domain Specific Languages
DSP	Digital Signal Processing
ECN	Explicit Congestion Notification
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
HEVC	High-Efficiency Video Coding
HM	HEVC Test Model
IDR	Instantaneous Decoding Refresh
IRAP	Intra Random Access Point
LAN	Local Area Network
MPEG	ISO/IEC Motion Picture Experts Group
NAL	Network Abstraction Layer
PDP	Programmable Data Plane
PSNR	Peak Signal-to-Noise Ratio
P4	Programming Protocol-Independent Packet Processors
QP	Quantization Parameter
SDN	Software-Defined Networking
VoD	Video on Demand
VCEG	Video Coding Experts Group
WAN	Wide Area Network

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Video streaming has been the main driving force of Internet traffic growth in the past few years, having accounted for over 60% of the global traffic in 2019 (Sandvine Inc., 2019) and almost 50% of mobile traffic in 2021 (Sandvine Inc., 2021). Many reports also predict that streaming will jump to 80% of global traffic share in the coming years (Sandvine Inc., 2019). This trend is explained by three main factors: (*i*) the higher number of connected multimedia devices, estimated to be more than three times the global population by 2023 (Cisco Inc., 2021); (*ii*) the increased use of high and ultra-high resolution videos, with more pixels than other standard video resolutions, resulting in increased Internet traffic, e.g., 66% of connected flat-panel TVs will support 4K (3840×2160 *pixels*) by 2023 (Cisco Inc., 2021); and (*iii*) the popularization of video streaming due to video-on-demand (VoD) services.

The growing demand for video streaming will pose ever-increasing stress on the global networking infrastructure, a situation only worsened with the coronavirus pandemic (KANG; ALBA; SATARIANO, 2020). It means that to keep up with users' expectations for high-quality streaming, networking researchers and practitioners will need to upgrade the networking infrastructure (e.g., with faster links) and devise solutions to optimize its usage (encoding schemes, delivery strategies, etc.). In this work, we focus on delivering strategies to optimize video streaming network usage.

One such strategy is processing video streaming packets as they transit the network (TONI; CHEUNG; FROSSARD, 2016; BAGCHI et al., 2019), by taking into account current network conditions (e.g., the proportion of users from a given network region interested in the streaming, available network bandwidth, etc.), thus complementing the decades-old rate-limiting video transmission strategy based on the end-user software feedback (COWAN et al., 1995). Nonetheless, network congestion and packet losses remain a significant challenge, and encoding schemes resilient to losses become paramount.

In this context, the bitstream in recent video coding standards such as H.264/AVC (Advanced Video Coding) (ITU-T; ISO/IEC, 2003) and H.265/HEVC (High-Efficiency Video Coding) (ITU-T; ISO/IEC, 2013) is partitioned into Network Abstraction Layer (NAL) units to facilitate video transmission over lossy packet-based networks (SJOBERG et al., 2012). Unlike previous standards, H.264/AVC and H.265/HEVC are robust to packet losses since video can be decoded even when some packets are lost. In HEVC this is done with the division of coded video data and metadata into NAL units of different

types, Intra Random Access Point (IRAP) and non-IRAP picture NAL units. The former contains video data encoded with intra prediction and is self-contained since it does not need other NAL units to be reconstructed. IRAP NAL units are particularly important in the decoding process, as they are used as a reference to reconstruct non-IRAP pictures that employ inter prediction through motion compensation.

Due to the error propagation in frame encoding, packet losses influence video quality more aggressively when IRAP NAL units are lost in error-prone packet-based networks. To the best of our knowledge, no solution has explored the idea of preemptively discarding non-IRAP packets under network congestion. One option to address the network congestion issue is to devise an in-network mechanism capable of monitoring the video stream and selectively discarding packets that pose less impact on the video quality. Recent advances in network programmability and computer architectures have allowed designers to move the computation inside the network, where it is closest to the data (CORDEIRO; MARQUES; GASPARY, 2017; SAPIO et al., 2017). Even though we cannot stop packet loss, we can analyze packets going through the network and prioritize data, such as IRAP over non-IRAP NAL units.

As a first step, we propose an in-network computing approach for selective video streaming packet discarding under network congestion. In our proposal, we monitor video streaming based on video coding standards such as H.264/AVC and H.265/HEVC and selectively discard non-IRAP picture NAL unit packets in a content-aware network flow. To this end, we rely on a hardware architecture that extends traditional packet-forwarding devices (e.g., switches) employing network programmability to analyze network flows, detect congestion, and preemptively discard non-IRAP NAL units. First, we evaluated the algorithm based on the percentage of IRAP packets it preserved at the end of a period of network congestion, which showed great results.

In the sequence, aiming to get closer to an analysis of how much packet loss would impact the end-user experience, we chose to analyze the relationship between packet loss and its consequent loss of video frames. Analyzing the relationship between the loss of IRAP packets and the consequent loss of frames, it can be seen that they are closely related, and the percentage of loss of frames is always slightly higher than the percentage of loss of packets in the same condition. One of the reasons why these two metrics are related is that when losing some IRAP packets, the IRAP frame and some subsequent frames that would be generated from it are lost.

As a future work, we intend to analyze our algorithm and hardware module in

terms of video quality based on Peak Signal-to-Noise Ratio (PSNR) metrics. With this approach, it is expected to better measure the relationship between IRAP packet losses and frame losses, previously verified, and the resulting video quality. In addition, working with more realistic network congestion simulations should bring our results closer to what the streaming end user perceives in terms of quality. Based on these metrics, adjustments to the originally proposed algorithm may be necessary, in addition to determining maximum acceptable limits for IRAP packet losses.

In the following chapters, a review will be made of the theoretical foundations that support this research (Chapter 2), the existing research related to Video Transmission and In-network Computing (Chapter 3), our proposed architecture (Chapter 4), the evaluation of this architecture (Chapter 5), our conclusions (Chapter 6) and future work (Chapter 7).

2 BACKGROUND

This chapter discusses the fundamental topics that underlie this work, including video streaming, video quality analysis, video compression, software-defined networks, domain-specific languages, and FPGA.

2.1 Video Transmission

Video is transmitted over the Internet in compressed form, often using international standard codecs developed by ISO/IEC Motion Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG). These standards define the decoding process and the syntax of the compressed video (bitstream). Real-time video transmission over the Internet in the presence of packet losses is not a new problem, and many techniques were proposed to make it more resilient to errors (WANG et al., 2000).

An important development in this subject was introduced in H.264/AVC standard (ITU-T; ISO/IEC, 2003). This standard was developed considering video transmission through packet-switching networks. The bitstream in H.264/AVC is partitioned into NAL units, separating video data and metadata and exposing some information to the transport layer. This is done through the NAL unit header, which includes a NAL unit type field to identify sequence and picture parameter sets (metadata information about video sequences and pictures), IDR pictures (Instantaneous Decoding Refresh - used for random access in the bitstream), non-IDR pictures, and so on. The H.265/HEVC standard (ITU-T; ISO/IEC, 2013) inherits this concept while defining the more general IRAP pictures/NAL (SJOBERG et al., 2012).

2.2 Video Quality

As video transmission becomes more present in users' lives and its consumption also occurs on high-resolution TV screens, user expectations for quality keep increasing. It was found that the percentage of time spent in buffering has the largest impact on the user's engagement across all types of video content, even more than the resolution/quality reduction (DOBRIAN et al., 2011). Understanding this impact is becoming more relevant every year so new technical alternatives are sought to avoid buffering time while main-

taining a sufficient quality of video displayed to the users.

Other studies explored the correlation between packet loss in a video streaming and resulting video quality (CHEN et al., 2011). It was concluded that the higher the packet loss rate, the worse the image quality, due to the possibility of losing more information. The reason for the proportional relevance between the packet loss rate and the size of the affected image area is that the decoding error caused by the packet loss would accumulate and influence other parts of the image. It has also been found that the compromised video quality is directly related to the frame type of the lost data. In the loss of data from the most important frames, there is a distinct mosaic. While in the less significant frame data loss image, there is ghosting around the edges in some parts of the image. Due to coding characteristics, loss of IRAP frame data would affect the decoding result of the IRAP frame and its subsequent frames, and cause mosaicing at the end. If there is data loss of other types, it means that some information such as the motion vector is missing. As a result, the missing data will affect the inter-frame prediction, leading to a jittery feeling when viewing the video.

2.3 Video Compression/Encoding

Compression is the act of compressing data into fewer bits than the original data. Video compression is the conversion of a digital video into a format that takes up less space, either to store or transmit it over a network, in a live video streaming, for example. The "RAW" format of a video requires a large and fixed bitrate, that is, all video frames occupy the same amount of memory, so compression is necessary to be able to store or transmit large amounts of video (RICHARDSON, 2011).

Video compression usually involves a CODEC pair (enCOder/DECoder), where the encoder is responsible for compressing the data before transmission or storage, while the decoder is responsible for decompressing the data to a representation of the original video. To perform the compression several techniques are used, mainly exploring redundancies in the stored data. We can divide the types of compression into two main types: lossless compression, where statistical redundancies are removed so that the reconstructed video is an exact copy of the original; and lossy compression, where subjective redundancies are removed, which do not affect the final result much but the resulting video will never be exactly like the original again.

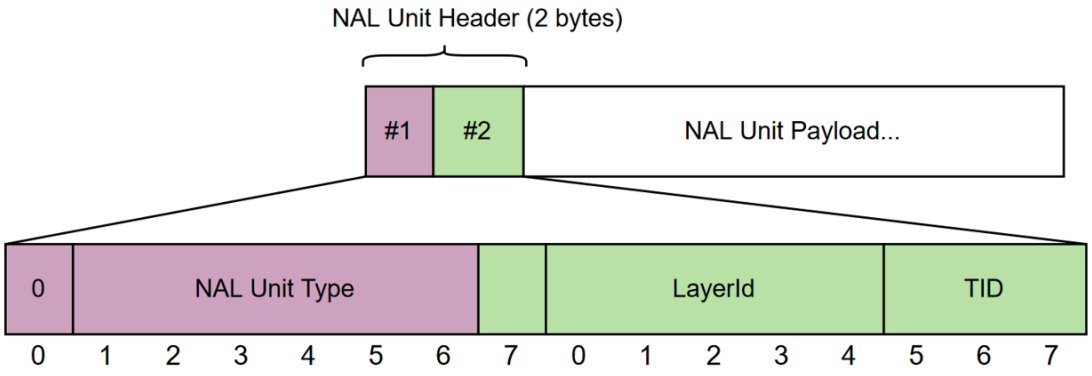
In video compression one of the parameters that can completely change the final

result is the Quantization Parameter (QP), in HEVC/H.265 the QP ranges from 0 to 51. The quantization parameter is used to determine the quantization step, which doubles each time QP increases its value by 6. Quantization is the act of mapping a set of values to a smaller set of values, it works using an integer division where the quantization step is the divisor and the original value is the dividend. So this process is irreversible because it introduces data loss and the higher the quantization parameter the stronger will be the quantization and consequently the data loss.

2.3.1 H.265/HEVC standard

The H.265/HEVC standard (High-Efficiency Video Coding), as its name said, is exceptionally good at compressing videos, about twice as much as H.264. An HEVC bitstream consists of a sequence of network abstraction layer (NAL) units, in Figure 2.1 it is possible to see that the first two bytes of a NAL unit belong to its header, and the rest is the payload. Using this header we can know the kind of picture stored in each NAL.

Figure 2.1 – NAL Header



Source: The Author

NAL unit types can be divided into IRAP pictures and non-IRAP pictures (SZE, 2014). In Table 2.1 we can see some of the different types of NAL units that exist, but in this work, we will focus just on IRAP and non-IRAP pictures. In HEVC each picture is partitioned into slices (one or multiple), each slice is part of a picture that can be decoded without using data from other slices in the same picture. At the same time, to follow network transmission restrictions each slice can be divided into multiple slice segments, where the second slice segment and the next ones are dependent on the first segment of the slice. It is important to note that each NAL unit only carries one slice segment.

The IRAP picture consists of a picture that is coded not using the content of other

Table 2.1 – NAL unit types

Description	Type code
Trailing non-IRAP pictures	0-5
Leading pictures	6-9
Reserved non-IRAP	10–15
Trailing non-IRAP pictures	16-21
Reserved IRAP	22–23
Reserved non-IRAP	24–31

Source: The Author

pictures as reference, they are used to set points in the bitstream where it is possible to start the decoding process. The first picture of a bitstream must always be of type IRAP, but there are IRAP pictures scattered throughout the bitstream. To store or transmit the bitstream, IRAP pictures are usually sent at regular intervals, with this it is possible to quickly provide random access to any part of the video since decoding can start from any IRAP picture. In real-time transmissions, where random access is not so useful, fewer IRAPs are sent and not periodically, only when corrupted data is noticed and a new IRAP is needed to refresh the video scene.

The non-IRAP pictures in a video are all the other pictures that depend on an IRAP picture to be completely decoded. There are a lot of types of non-IRAP pictures, each one with its function, but in essence, they are pictures that are usually associated with the closest IRAP picture in the bitstream and because of this, does not need to store all the frame, just some information about the difference between the current frame and the frame of the associated IRAP picture.

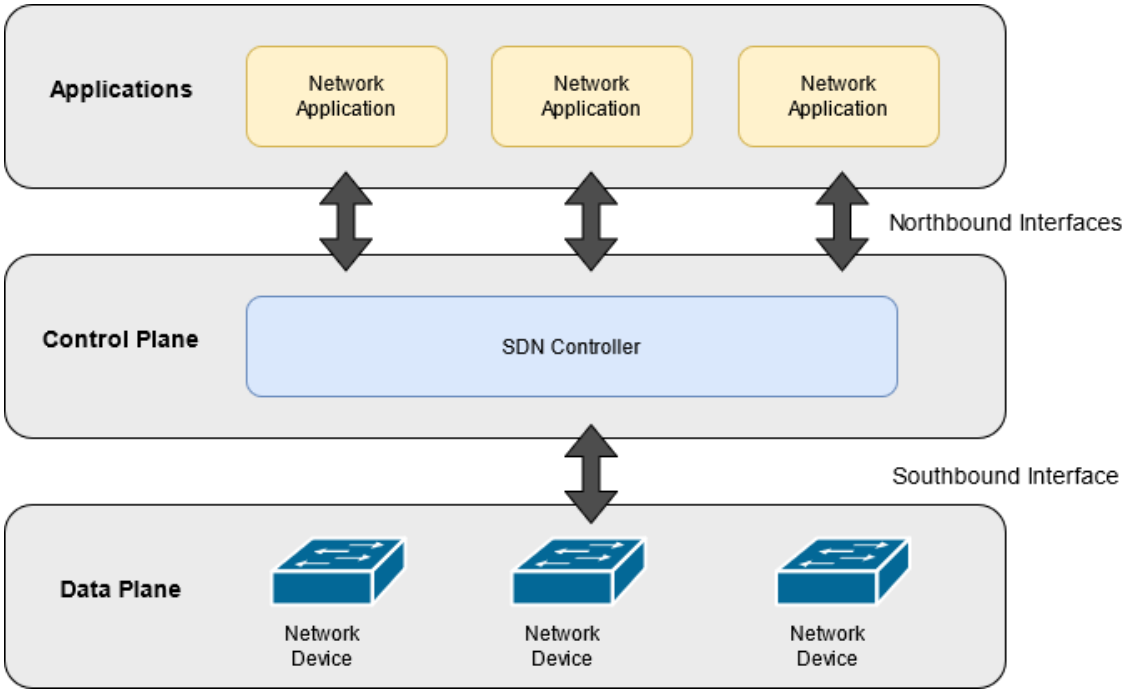
2.4 Software Defined Networks

The emergence of Software-Defined Networking (SDN) has allowed researchers and network operators to think about the idea of a more programmable network. With SDN, the network intelligence is decoupled into a control plane and a data plane to make the network more flexible, taking the control logic out of the network devices, and letting it in charge of the forwarding roles. Fig. 2.2 shows an overview of an SDN architecture.

Applications are in a higher abstraction than the control plane and interact with it through northbound interfaces. In SDN, the control plane becomes responsible for specifying forwarding rules, such as specified through the use of an SDN controller and an interface (southbound) for managing the table entries of network devices operating on

the data plane. This way, SDN becomes a vehicle that can make it possible for network operators to manage network devices in a more centralized manner since the configuration of network devices is focused on a single entity.

Figure 2.2 – A overview of the SDN architecture.



Source: The Author

2.5 Domain Specific Languages

A domain-specific language is a computer programming language of limited expressiveness, focused on a specific domain. There are four key elements in this definition: (i) Computer programming language: A DSL is used by humans to instruct a computer to do something. As in any modern programming language, its structure is designed to facilitate understanding by human beings, but it must still be something executable by a computer; (ii) Natural language: a DSL is a programming language and, as such, must have a sense of fluency in which expressiveness comes not only from individual expressions but also from the way they can be composed; (iii) Limited expressiveness: a general-purpose programming language provides many features, such as support for varied data structures, control, and abstraction. All of this is useful but makes learning and using difficult. A DSL supports the minimum resources required to support your domain. One cannot build an entire software system on a DSL; Rather, you use DSL for a specific aspect of a system; and (iv) domain focus: a limited language is only useful if it has a

clear focus on a small domain. The focus of the domain is what makes a limited language worthwhile (FOWLER, 2010).

The adoption of a domain-specific language involves benefits and disadvantages, and working with this approach means finding a balance between them. The benefits of DSLs include: (i) DSLs allow solutions to be expressed in the language and level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify, and often even develop DSL programs (DEURSEN; KLINT; VISSER, 2000); (ii) The programs are concise, largely self-documenting, and can be reused for different purposes (LADD; RAMMING, 1994); (iii) DSLs increase productivity, reliability and maintainability (KIEBURTZ et al., 1996) and allow validation and optimization at the domain level (BRUCE, 1997); and (iv) DSLs improve testability and maintainability (SIRER; BERSHAD, 1999).

However, according to (DEURSEN; KLINT; VISSER, 2000), the disadvantages of using a domain-specific language are (i) The costs of designing, implementing, and maintaining a DSL; (ii) The costs of education for DSL users; (iii) The difficulty of finding the appropriate scope for a DSL; (iv) The difficulty of balancing between domain specificity and general-purpose programming language constructions; (v) The potential loss of efficiency when compared to manually coded software; and (vi) the limited availability of DSLs (KRUEGER, 1992).

2.5.1 Programming Protocol-Independent Packet Processors

Programming Protocol-Independent Packet Processors (P4) is a high-level language that provides a suitable abstraction model for Programmable Data Plane with the capability of specifying and programming the data plane behavior of a network device. (BOSSHART et al., 2014). Its main objectives are reconfigurability, protocol independence, and target independence. Reconfigurability proposes that the analysis and processing of packets can be redefined by the controller. Protocol independence suggests that the controller can extract header fields by specifying the packet analyzer and a collection of match-action tables for header processing. The target independence focuses on the possibility of the network operator specify the behavior of the switch while abstracts the non-relevant information about the target device.

A P4 program mainly defines six items: (i) headers, which specify the names and widths of the protocol fields in which the program operates; (ii) metadata, structures that

provide specific package information; (iii) parser, a group of state machines for analyzing headers and extracting data for metadata structures; (iv) match-action table, which identifies fields and metadata of packages to be compared and the possible actions being taken in response; (v) execution pipeline and control flow, to define how the packages are processed; and (vi) deparser, a process for rebuilding packages.

The task flow of the definition of a network device with P4 support begins with network operators writing and compiling a P4 program using a front-end compiler in a High-Level Intermediate Representation (HLIR). Then, a back-end compiler adapts the program to different targets, including FPGAs. Finally, through a runtime controller, operators can populate entries in the match-action tables in the data plane and allow the destination device to process and forward packets.

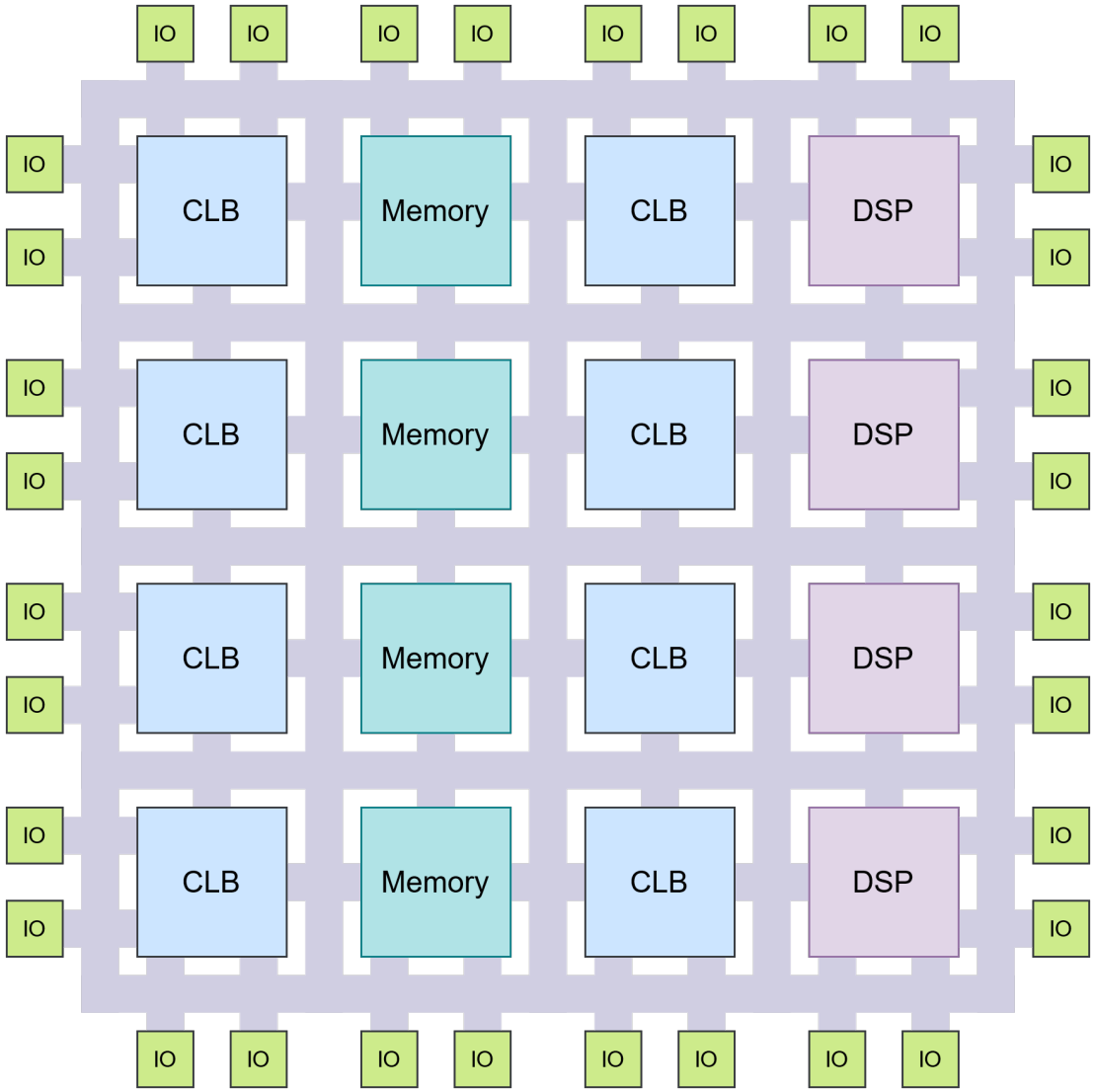
2.6 Field-Programmable Gate Arrays

The use of hardware-accelerated network devices in switches and routers is enabling rapid growth of the Internet. Currently, Ethernet switches are widely used to switch traffic on Local Area Networks (LANs) and to route protocol packets across Wide Area Networks (WANs). Commercial vendors use ASICs and/or FPGAs to speed up the switching, routing and processing of network data (LOCKWOOD et al., 2007). As previously explained, the new generation of SDN-related solutions introduced the notion of programmability of the data plane (for instance, P4 and POF). They allow faster development/provisioning of new protocols, as opposed to the long wait for the release of fixed-function ASIC switches that support standardized protocols (SIVARAMAN et al., 2015). The intrinsic characteristics of the FPGAs are aligned with the programming of the data plane that seeks reconfigurability and programmability.

More specifically, FPGAs are prefabricated silicon devices that can be electrically programmed to become virtually any type of digital circuit or system (KUON et al., 2008). They offer several attractive advantages over fixed-function ASICs (CHINNERY; KEUTZER, 2002). In addition, ASICs take months to manufacture and cost hundreds of thousands to millions of dollars to obtain the first device, while FPGAs are set up in minutes (and can usually be reconfigured if an error is made in generating a configuration file, commonly called bitstream) and cost from a few dollars to a few thousand dollars. The flexible nature of FPGAs, however, has a significant cost in area, delay and energy consumption: an FPGA requires approximately 20 to 35 times more area than an ASIC, it

has a speed performance approximately 3 to 4 times slower than an ASIC and consumes approximately 10 times more energy (KUON; ROSE, 2007). These disadvantages arise in large part from the programmable routing mesh of an FPGA, which has a high cost in area, speed and power consumption to traffic data when compared to an ASIC.

Figure 2.3 – Basic internal organization of FPGAs.



Source: The Author

An FPGA is composed of fundamental building blocks called Configurable Logic Blocks (CLBs), which provide the physical support for a design to be implemented and loaded into the FPGA, also, it contains Digital Signal Processing (DSP) slices and memory blocks. Fig. 2.3 illustrates the basic internal organization of an FPGA. The CLB is the main logical feature of the FPGA to implement sequential, combinatorial, and logical functions. Each CLB is connected to the routing loop through a routing matrix (CHANDRAKAR; GAITONDE; BAUER, 2015). The DSP is designed to carry out digital signal

processing functions, like multipliers, in a more efficient way than implementing those functionalities with CLBs. The referenced memory itself is a block RAM. The use of FPGAs allows fast prototyping but also takes advantage of flexibility, high performance, and reconfiguration. The main difference in the use of FPGAs, when compared to the use of microprocessors, is the ability to make substantial changes to the hardware, including changes in data and control flows (YANG et al., 2014), in addition to better efficiency in terms of performance and power consumption, since it implements a dedicated circuit. When compared to using a custom ASIC, the advantage of using FPGAs is the possibility to change the hardware at runtime, loading a different circuit in the reconfigurable matrix.

2.7 P4vBox

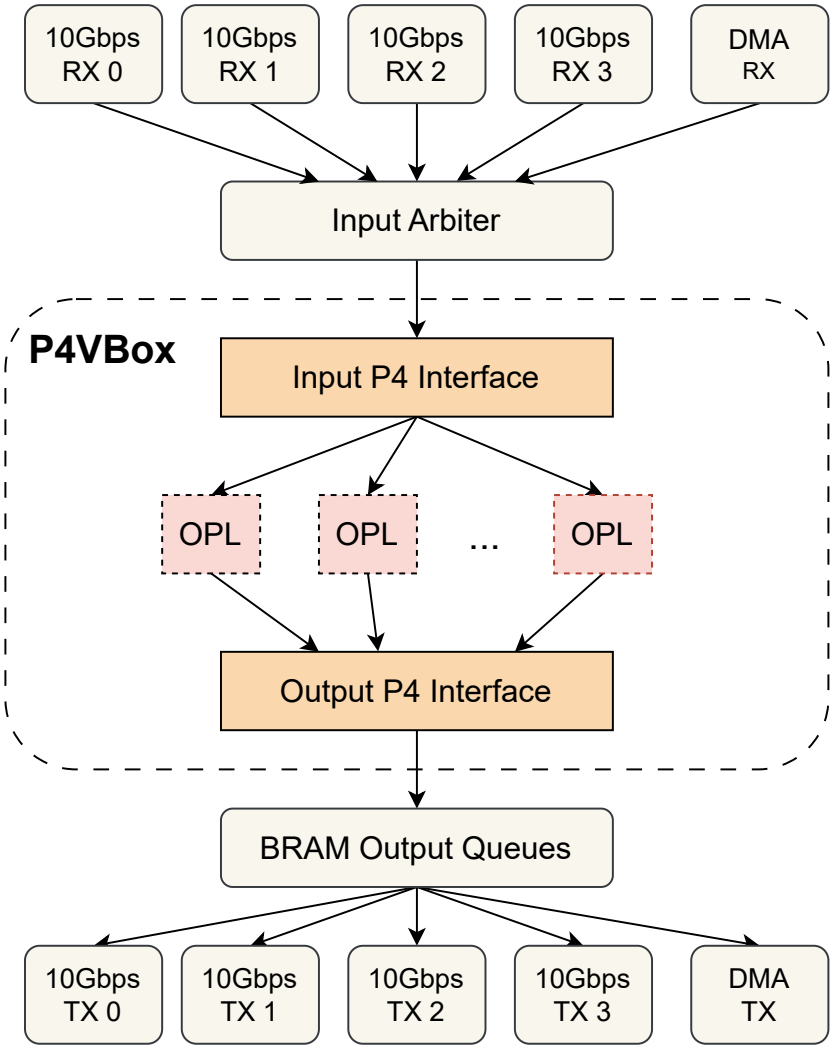
P4VBox (SAQUETTI et al., 2020) is an architecture that allows the virtualization of the data plane, using programmable switches, in a FPGA board. It proposes that virtualization be done without adding difficulties for the network operator, allowing him to deploy the same byte codes as a switch, both in the virtualized environment of the P4VBox and in another non-virtualized environment. In addition, it allows hot-swapping, which is the functionality of being able to change only one switch instance without affecting the others with zero downtime, this is done using the partial reconfiguration functionality.

In Figure 2.4 we can see the P4VBox design, the whole architecture has four 10Gbps and Direct Memory Access (DMA) I/O ports, BRAM Output Queues, and an Input Arbiter. P4VBox specifically has the Input P4 Interface, the Output P4 Interface, and allows multiple Output Port Lookup (OPL) instances. Each OPL instance, in this case, is a different and completely isolated switch instance, all these switches can be written in P4 and generated using the High-Level Synthesis workflow.

In order to deal with the multiples switches receiving flows from multiple ports, P4VBox implements the Input P4 Interface that uses a VLAN (802.1Q) concatenated with the frame destination address to identify which switch should receive each packet. The Output P4 Interface has a similar behavior, but works to deliver the packets to the right output port at the end of the processing, working similar as a demultiplexer.

To use the P4VBox, you must program the network device using P4, pass it through a High-Level Synthesis flow, and finally, optionally, add some hardware modules written in Verilog. The P4VBox allows the network operator to deploy switches up to two orders of magnitude faster than its competitors, making it very good for prototyp-

Figure 2.4 – P4VBox Design



Source: The Author. Adapted from (SAQUETTI et al., 2020)

ing and testing. In addition, with the set of commands and options it implements, using its recommended flow, it is possible to quickly and practically obtain results of latency and occupation of programmed devices.

3 RELATED WORK

This chapter presents techniques used to improve the quality of video transmitted over computer networks. In addition, types of hardware accelerators that can be used in a network to optimize video transmission are presented.

3.1 Video

This section aims to present different techniques that analyze and try to improve the quality of video transmissions over the network. They work with aspects ranging from analyzing the impact of degraded videos to techniques for modifying the video bitstream to make it more resilient in a packet loss condition.

The authors from (KORHONEN, 2018) used H.264/AVC to present a subjective video degradation analysis under packet loss scenarios while differentiating the impact of reference and non-reference pictures. In their studies, they explored the impact of video artifacts using a touchscreen where video sequences were displayed and the users tapped into the positions where they found some artifacts. They also analyzed the features derived from those artifacts and proposed two models for estimating and combining those features into an objective metric for assessing the apparency of the artifacts. One model is just analytical and the other one is learning-based. Using these two models they found out that multiple different factors can influence the visibility of packet loss artifacts, and expected that by knowing better these factors, researchers can develop new methods to minimize the impact of packet losses in video transmission.

In (KAZEMI; IQBAL; SHIRMOHAMMADI, 2018), an encoding-time technique is presented for defining intra-predicted frames (i.e., IRAP) positioning in the H.264/AVC bitstream to improve packet loss resiliency. Multiple description coding (MDC) is a technique for video transmission over error-prone networks that fragments a single media stream into n substreams, so in this paper, they worked on a new intra-coding approach in MDC. They found out that using MDC streams, the best policy is to encode some kinds of frames as IRAP ones instead of encoding some parts of other frames in intra mode. Using this new encoding technique they could achieve higher PSNR compared to other MDC techniques, however, their technique degrades the compression ratio.

A similar strategy is adopted in (WALLENDAEL et al., 2021) where IRAP frames (referred to as keyframes by the authors) are periodically inserted during the encoding

process to improve H.265/HEVC bitstreams resiliency. They made this work by using both a compression-efficient video stream with another stream consisting of just IRAP frames. Using some restrictions and modifications they were able to make this work with video encoded by H.265/HEVC, providing a quicker error recovery at low-quality impact. Even though the inserted keyframes are 7 to 30 times greater than the replaced frames, the quality decrease is smaller and less perceptible.

The work in (OZTAS et al., 2012) analyzes the behavior of HEVC and AVC over many different scenarios and concludes that HEVC is less error resilient than H.264/AVC, especially in videos with a high amount of motion. Is important to note that the HEVC's higher level of compression compromises the error resilience reducing significantly the necessary bandwidth. In more static scenes the HEVC becomes more resilient due to the amount of similarity between two frames, which works like a frame coping. They also found that HEVC handles transmission errors better when the resolution is higher because each frame needs more packets to be transmitted and each packet carries a smaller portion of the frame, so losing a packet has not have a high impact on the video quality.

The work in (NIGHTINGALE; WANG; GRECOS, 2012) has developed a framework for streaming and evaluating HEVC-encoded video over lossy networks. These lossy networks were simulated in a realistic testbed, with external interference, bandwidth limitations, and even packet loss. The framework provides an error concealing method to overcome some limitations of HEVC, making changes both in the decoder and in the way NAL units are divided into packets. These modifications are carried out in three main stages. In the first one, encoding and prioritization are carried out simultaneously, where the modified log of the first one provides parameters for the second one. The second stage, where the bitstream is extracted and packetized, based on previously defined priorities and subsequent streaming; and finally, the correction/hiding of errors, where when perceiving a frame that the decoder could not decode, the framework copies the closest frame. After all these steps, the quality analysis is performed. As video quality results are usually measured in peak signal-to-noise ratio (PSNR) in past works, they chose to keep the same metric in this one. Measuring the PSNR they reach an average loss of 3.61dB when reducing the bandwidth by 10%.

3.2 Network Hardware and Accelerators

In the context of computer networks, many advances have occurred in the past decade, especially in terms of Software-Defined Networking (SDN). Such advances have led to the emergence of network programmability (BOSSHART et al., 2013; CORDEIRO; MARQUES; GASPARY, 2017) and have provided network administrators with the ability to reprogram the behavior of forwarding devices through Domain-Specific Languages (DSL) such as P4 (BOSSHART et al., 2014). In the same way that the networking infrastructure advanced in programmability, it also advanced in computational power. With new programmable network hardware and accelerators on the market, such as Smart Network Interface Cards (SmartNICs) (SANVITO; SIRACUSANO; BIFULCO, 2018), Graphics Processing Units (GPUs) (SUN et al., 2019), and FPGAs (WOODRUFF; RAMANUJAM; ZILBERMAN, 2019), a new generation of programmable network devices is flooding the market, enabling computation to be performed within the network.

In (SANVITO; SIRACUSANO; BIFULCO, 2018) they studied the viability for using programmable network devices, such as SmartNICs, as a Artificial Neural Networks accelerators. A network interface card is a component that connects the computer or server on which it is installed to the internet. A SmartNIC is a network interface card that has extra pieces of hardware dedicated to performing functions such as storing and processing security functions. SmartNIC are generally composed of a standard network interface with a CPU or an FPGA attached. The ability for functions to be executed on the SmartNIC reduces the processing load on the main processor of the machine where the SmartNIC is installed, which is very useful on servers with high bandwidth.

Graphics Processing Units (GPUs) are devices that are also widely used on servers and very common in High-Performance Computing (HPC) applications. General-Purpose GPUs (GPGPUs), more specifically, transcend the domain of computer graphics and can be used to accelerate hardware in areas such as deep learning and cryptocurrency mining. Although GPUs are efficient for running heavy computing applications, they are usually not directly connected to the network, which makes them less suitable for network-intensive applications (TOKUSASHI et al., 2019). It is believed that this is one of the reasons that the in-network computing literature does not present many works that make use of these devices, however, it is difficult to assess how far network computing can go and what are the possibilities that the next generations of hardware will make available.

There are a lot of studies that worked with FPGA-based Hardware Accelerators

using In-Network Computing. The work of Cooke and Fahmy (COOKE; FAHMY, 2020) developed a model for evaluating the different approaches of in-network computing, that consider: the multiple levels of network structure; hardware differences and its changes on computing and networking, which includes accelerator platforms; realistic representation of metrics, such as performance, energy cost and financial cost. In order to test the proposed model they develop some Python scripts. This model is used to investigate a case-study scenario that demonstrates reductions in the latency of communication between devices, in addition to low computation latency when using acceleration in FPGA. An image classification application based on neural networks called SqueezeNet is used as a case study and quantifies the computation latency resulting from different platforms.

The NetDebug (BRESSANA; ZILBERMAN; SOULÉ, 2018) proposes a fully programmable hardware-software framework for real-time validation and debugging of programmable data planes at full line rate. The NetDebug prototype was built on a NetFPGA-SUME using Xilinx SDNet, which translates P4 descriptions into hardware modules. Also, it is independent of the language of the determined application, validating data planes even in such a different workflow like high-level synthesis. However, the first assessment found that the state of the reject parser, an essential feature of the P4 language, is not currently implemented by SDNet compiler.

LaKe (TOKUSASHI; MATSUTANI; ZILBERMAN, 2018) is a layered key-value storage design, executed as a network application. It works using multiple layers of cache, where each layer provides a trade-off between memory size and performance. This FPGA-based proposal achieves 17$\times$ better power efficiency than running on a host, with a transfer rate of 6.7$\times$ up to 13.6$\times$ higher, maintaining two orders of magnitude with better latency. Lake was implemented using Verilog, a hardware description language, on a NetFPGA-SUME board, but this language restriction can be impeditive for a programmer.

iSwitch (LI et al., 2019) proposes a distributed solution using in-network computing to move gradient aggregation operations from network node servers to FPGA-based switches, reducing the number of network hops during gradient aggregation operations. Gradient aggregation is Reinforcement Learning (RL) operations used to train Artificial Intelligence applications. The results demonstrate that compared to the latest generation distributed training approaches, iSwitch offers an acceleration of up to 3.66 times for synchronous and 3.71 times for asynchronous distributed training while achieving better scalability. Also, the authors rethought the distributed RL training algorithms and propose a different hierarchical aggregation mechanism to increase parallelism and scalability.

3.3 In-network Computing

In-network computing is still in its infancy, with most works targeting networking-related applications, such as caching (MATSUZONO; ASAEDA; TURLETTI, 2017), and data aggregation (SAPIO et al., 2017), with only a few works aiming at other areas such as artificial intelligence (LI et al., 2019; SAQUETTI et al., 2021) and adaptive video rates (BRONZINO et al., 2014; BOUTEN et al., 2014).

In the (MATSUZONO; ASAEDA; TURLETTI, 2017) work, a mechanism was proposed that reduces losses and latency in video transmission using an extension made for Content-Centric Networking (CCN), which basically allows data consumers and routers to use network caching, contributing to a better use of computer network resources. The work is based on the fact that, given a network with a high volume of data sensitive to delays, each node makes an estimate of acceptable delay and probability of loss, in order to apply the techniques according to these values.

The applied techniques are a mixture of using multiple paths in the network, using efficient technologies for this, such as the cache for retransmission of lost packets, video coding in the network nodes, and adapting the flow in the network. In addition, in some specific cases of competition for bandwidth, changing encoder parameters to adjust the quality of the video produced and its consequent size. It was validated that this solution behaves better than the standard use of a CCN, being an interesting solution that makes use of In-network computing to optimize video transmission.

The MobilityFirst (BRONZINO et al., 2014) project proposes the extension of the data plane with attached modules. This extensions modules can improve the experience of end-users of mobile devices while sending some part of the workload to a network device. They present also a new protocol and API extension to increase the flexibility of the solution usage. Related to video streaming their solution used the data from all nodes to estimate the available bandwidth to calculate the best bit rate of the video, adjusting at execution time the compression settings.

4 PROPOSED IN-NETWORK COMPUTING ARCHITECTURE

The proposed in-network computing architecture discussed in this work targets a generic packet-forwarding device. It aims at decreasing IRAP packet loss during network congestion while maintaining packet loss to a minimum. To do so, we conceived an in-network hardware module that implements a content-aware preemptive non-IRAP packet drop algorithm. This hardware module implementation could prove the feasibility of our approach in terms of resource usage and performance, having great results with a measly cost on resources without interfering with the main device latency. In order to test and develop the algorithm we also created a fully customizable simulator that allows us to quickly test between algorithms. In the following sections, we describe in detail the chosen packet drop algorithm, how we developed the in-network hardware module that implements the algorithm, and the simulator.

4.1 Proposed Packet Drop Algorithm

Because IRAP frames are whole figures within a video and are used to reconstruct the other frames adjacent to it in the decoding process and the loss of one of them can cause several frames to fail, the main objective of our Proposed packet discard algorithm is to minimize IRAP packet loss. This idea aims to try as much as possible not to drop the IRAP packets so that it is possible to reconstruct each IRAP frame from the video that is been transmitted over the congested network.

A more naive option would be to, as soon as the network congestion was detected, start preemptively drop any non-IRAP packets until the congestion is gone. However, this can lead to the unnecessary loss of non-IRAP packets, as depending on the case the buffer may be able to store all the IRAP packets forwarded during the congestion and even additional non-IRAP packets. Thus, a better option would be to evaluate how many packets we can store during the congestion based on packet throughput and buffer occupation. By doing so, we can guarantee that our proposed packet drop algorithm will not unnecessarily drop non-IRAP packets.

We chose an algorithm that, upon detecting network congestion, manages the free space in the device's buffer. This management is based on the fact that, given an occupation of X in the buffer and a ranking of importance among the video packets, the algorithm is able to suggest to the device on which it is installed whether or not to drop each packet

that passes through it. during the period of network congestion.

By previously knowing the size of the device's buffer and its occupation in real-time, the algorithm is able to decide with greater confidence whether or not to drop the packet. This control is done in order to avoid losing IRAP packets at all costs. For example, at a given moment when the buffer has free space for 30 packets, but the algorithm estimates that there are about 50 packets left in this congestion period, the algorithm will suggest the drop of any non-IRAP packet. In another moment, when the buffer has free space for 25 packets and it is estimated to receive another 15 packets during the congestion period, then the algorithm will not suggest a drop for any packet. In this way, the algorithm tries to protect the IRAP packets at all times when it considers that there is a danger of losing them if the congestion persists. One of the reasons for choosing this algorithm to start the studies is due to the ease of detecting whether a packet contains part of an IRAP frame or not, and thus testing the viability of the original idea of improving video quality during network congestion.

Algorithms 1 and 2 describe the pseudocode of our packet drop routine, both algorithms are called for each packet that passes through the network device. Algorithm 1 is the one responsible for triggering a timer, which puts the system into congestion mode, whenever it receives a packet with a congestion notification. In this experiment the timer was set to 20, representing packets arriving during the congestion period. This value was arbitrary and takes into account the device, its throughput, and the size of its buffer. This action is performed attributing the (c_time) constant to the ($timer$) variable, this (c_time) is predefined by the programmer and its value can be achieved by experimentation. This behavior of attributing (c_time) to ($timer$) every time a flagged packet is received works as if the timer was reinitialized while the congestion remains. This congestion notification could be received in flagged packets, for example through the classic TCP Explicit Congestion Notification (ECN) (FLOYD, 1994) (p_ecn). To do this work we consider that there is a separate function running in parallel to count the timer down to zero.

Algorithm 1 Timer Policy

```
1: function SET_TIMER(packet)
2:     if is_tcp_ecn_enabled(packet) then
3:         timer ← c_time
4:     end if
5: end function
```

Algorithm 2 is also called for each packet received. If still in congestion ($timer > 0$), we need to compute how many packets could be received until the end of the expected congestion window ($packets$) by multiplying the current throughput (p_tp) by the re-

maining timer ($timer$). In case the buffer can no longer hold any further packets to be received during the congestion event ($b\_free < packets$) and the packet is a non-IRAP ($!p\_irap$), our algorithm recommends a packet drop ($return\ true$). Otherwise, it recommends not dropping the packet ($return\ false$). Note that this is only a recommendation to the packet-forwarding device, as the decision to drop packets is the forwarding device's.

Algorithm 2 Drop Packet Policy

```
1: function DROP(b_free, p_tp, p_irap)
2:     if timer > 0 then
3:         packets ← p_tp × timer
4:         if b_free < packets and !p_irap then
5:             return true
6:         end if
7:     end if
8:     return false
9: end function
```

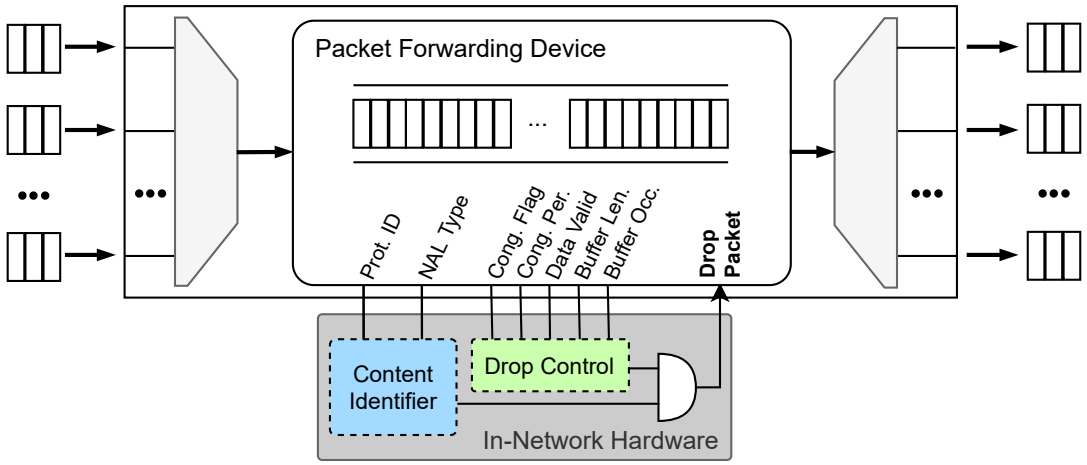
4.2 Proposed In-Network Hardware Module

Based on the algorithm previously defined, we developed an in-network hardware module that extends a generic packet-forwarding device through a simple interface. This implementation was made in FPGA given its flexibility, programmability, and agility in the implementation, and because we already have the necessary tools to implement a switch and a module attached to it (SAQUETTI et al., 2020). Using the P4VBox architecture, it is possible to, using a small set of commands, generate detailed latency and resource occupation reports.

As shown in Fig. 4.1, the in-network hardware implements our proposed algorithm with two blocks, the Content Identifier and the Drop Control. The Content Identifier receives a protocol identifier (*Prot. ID*) and the NAL type (*NAL Type*). The Drop Control block inputs information on the congestion: congestion flag (*Cong. Flag*) and congestion period (*Cong. Per.*), the validity of the data received (*Data Valid*), and information on the buffer status, length (*Buffer Len.*) and current occupation (*Buffer Occ.*), returning true if there is not enough space left in the buffer until the end of the network congestion. When both these flags are true, the in-network hardware module indicates to the packet-forwarding device that the packet should be dropped.

Regarding the Algorithm 1 our hardware module implements it as a register for the timer and a block for verifying at each clock if there is a new packet header with the flag on the data bus. We use this same block to, once the timer was set, count it down

Figure 4.1 – In-network Hardware Module Diagram



Source: The Author

by one for each received packet. In order to implement the Algorithm 2 we use the same timer register cited before, a new register for storing the remaining packets estimate and some data inputs, so it can determine the buffer state and if it is a good idea to indicate if the packet should be dropped. To get some of this information needed by the module we made use of another benefit of using P4VBox architecture because we have access to its source code it is possible to know exactly in which data bus the packets will flow and use our Verilog module to access it to get what we want. Also, this knowledge is useful to send buffer information across the packet's metadata. This section of the module is the most affected by algorithm changes since it has all logic behind the drop suggestion.

We implemented a layer-2 switch (l2-switch) as a baseline packet-forwarding device in P4 (BOSSHART et al., 2014) and our proposed in-network hardware module in Verilog. To interface both modules, we modified the l2-switch pipeline (in P4) to detect NAL headers and forward them through an external module to our hardware module (in Verilog). We then synthesized both projects (l2-switch and l2-switch extended) to the NetFPGA-SUME board with the Xilinx SDNet high-level synthesis tool, following the P4VBox (SAQUETTI et al., 2020) workflow. Our resource usage and performance evaluations were performed on Xilinx Vivado 2018.2 by injecting custom NAL packets.

Table 4.1 shows resource usage and performance of a case-study l2-switch and the same l2-switch with our extended in-network hardware module for packet dropping obtained in Xilinx Vivado for a NetFPGA-SUME. In terms of resource usage, our implementation required additional 2.6% LUTs and 1.9% FFs. Note that the l2-switch is one of the simplest packet-forwarding devices, and our implementation is agnostic to the forwarding device. Therefore, more complex devices would perceive an even smaller impact

on resource usage. Considering performance, the extended l2-switch achieved the same latency and throughput as the baseline, without degradation.

Table 4.1 – In-network hardware occupation

Resource and Performance	L2-switch	L2-switch extended	Overhead (%)
LUTs	1,774	1,821	47 (2.6%)
FFs	3,831	3,904	73 (1.9%)
Latency (μs)	0.56	0.56	–
Throughput ($Gbps$)	101.1	101.1	–

Source: The Author

4.3 Simulator

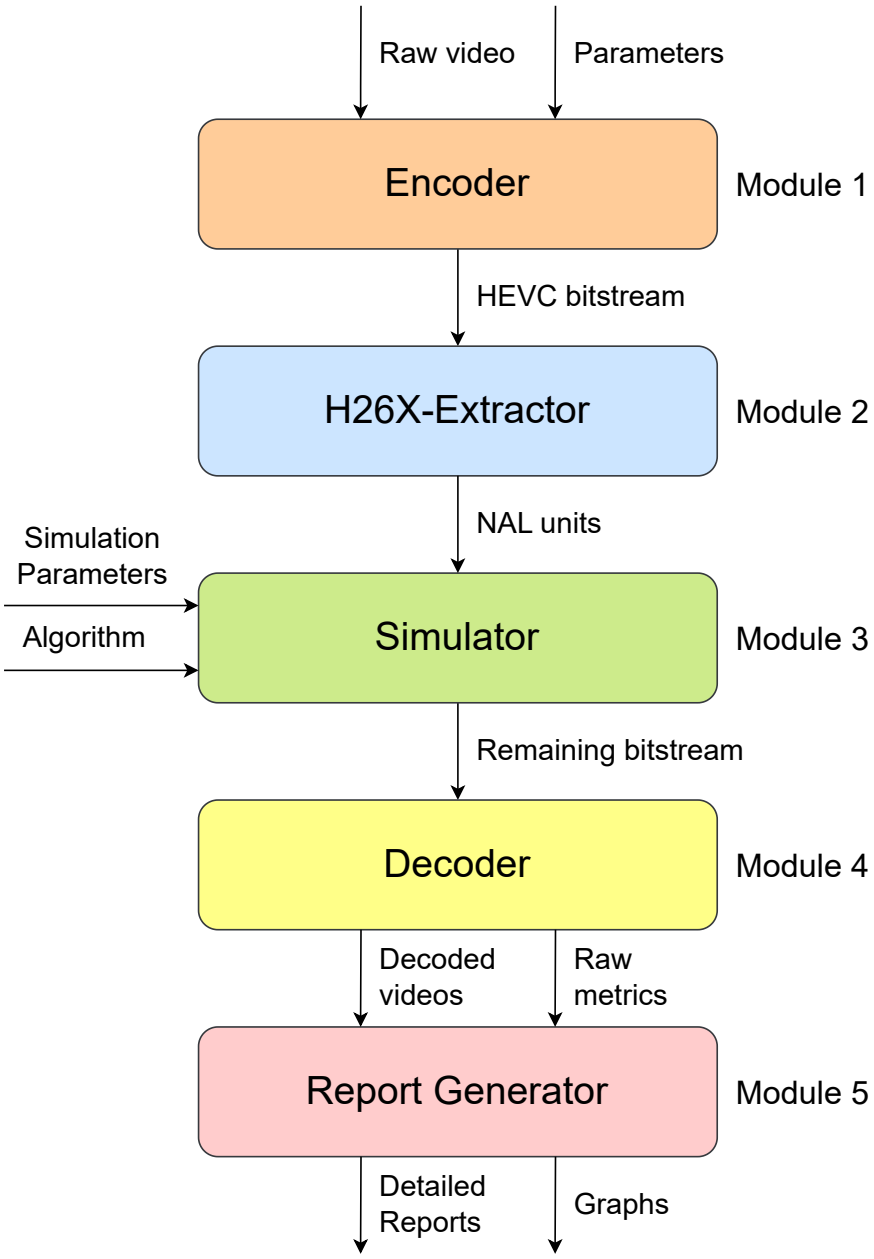
Since programming the algorithms in Verilog and testing hardware is more complex and time-consuming than writing the algorithm in a high-level programming language and the assemble and usage of a network testbed can take a lot of time we developed a simulator for the packet-forwarding device, which included tools for analysis of H.265/HEVC bitstream flows under congestion conditions. Using this simulator we were able to compare algorithms and just really implement the better ones. The simulation flow has 5 independent modules that need to be executed to obtain the final results. This modularity was done bearing in mind that other studies may only want to change, for example, the encoder used or the way congestion is simulated.

To make the analysis of packet drop algorithms easier, it was decided to develop this simulator using the Python 3 programming language and some bash scripts to carry out the steps using multiprocessing techniques. Python 3 was chosen due to its ease and speed of implementation and code maintenance, in addition to the great availability of external libraries to generate packets, handle the generated data and create reports, allowing the algorithm to evolve quickly.

In Figure 4.2 we can see the modules of the simulator. The first module is responsible for encoding the video to the H.265/HEVC format. For this, the HEVC Test Model(HM) reference software for HEVC in version 17 (ITU-T; ISO/IEC, 2023) was used. With it, it is possible to define files with configuration parameters similar that the ones used to validate other experiments around the world, this allows our results to be better compared with other works that pursue similar goals.

The second module is responsible for extracting the video data into a format that

Figure 4.2 – Simulator modules



Source: The Author

the simulator understands, for this purpose *h26x-extractor* (ROBITZA, 2021) open-source project was used, with an expansion to work with H.265/HEVC bit streams, where the output is a file with details of each NAL unit of the corresponding video. These details range from its header, with information about its type and consequent importance, to the payload (used to reconstruct the video after packet loss).

The third module is the simulator itself, it has a configurable test bench, allowing users to adjust packet input and output rates, buffer size, and congestion frequency and duration (where the output rate would drop to zero during the congestion). It is possible to

analyze relevant information about NAL units, such as the number of NAL units, packets per NAL, bytes per NAL, and percentages of NAL types, among other relevant data. The package drop algorithm is also implemented within the simulator, for the results of this work the Algorithm 2 was used, but given its modularity, it could be any algorithm that is based on the NAL type and returns the suggestion of drop or not of the package.

The fourth module is responsible for decoding the video using only the remaining packets and thereby obtaining metrics on how good or bad the algorithm is in terms of packet loss, loss of bytes, or even loss of frames. For decoding we again used the HM reference software decoder. This module is closely linked to the fifth module, which condenses all the information resulting from the previous module with some details obtained from the original video files and returns various reports with the requested information.

An important point to be emphasized is that, as the encoding, simulation, and decoding of each video are independent, all modules come with the option of running in parallel to optimize the time spent on the whole simulation, which is important to use since testing all cases addressed in this work get us a thousand different simulations. Tests were carried out on computers with 4, 8, and 20 cores, where a huge time advantage can be seen compared to running all simulations sequentially.

5 EVALUATION

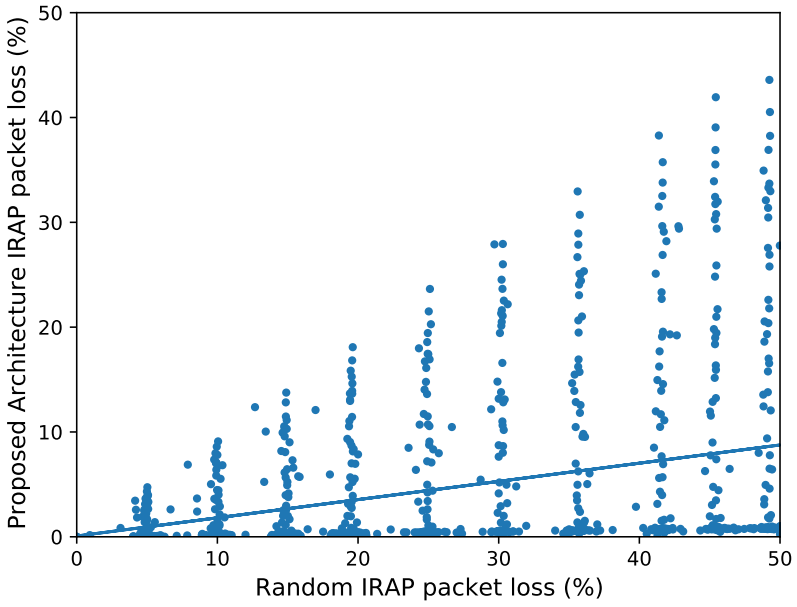
For our evaluation, we transmitted a set of videos through a simulated packet-forwarding device in congested network environments. We configured our simulator with a buffer size of 60 packets, a realistic value considering the target board used for synthesizing our proposed in-network hardware module. Considering that forwarding packet devices usually operate at line rate, having buffers specifically for cases of network instability, and for simplicity, we set the packet input and output rates at 60 and 120 packets per time unit, respectively. By doing so, the buffer will start filling as soon as the congestion starts and will be cleared as soon as the congestion ends. Additionally, we can manage packet loss by adjusting the frequency and duration of the congestion scenarios. To adjust our experiments for packet losses from 5% to 50%, in a 5% step, we varied congestion times from 10% to 94.11% of the simulation time.

As case-study video benchmark, we used all 25 videos from the Common Test Conditions (CTC) of H.265/HEVC (BOSSEN, 2013) encoded using x265 with the four recommended Quantization Parameters (QP - 22, 27, 32, and 37). The CTC contains videos with multiple resolutions (416×240, 832×480, 1024×768, 1280×720, 1920×1080, and 2560×1600), including different motion and texture characteristics. Considering that the videos are relatively short, we chose to concatenate them 100 times before sending them through our simulator, to reach statistical relevance ($\sigma = 0.0068$).

Fig. 5.1 correlates the random IRAP packet loss (X-axis), simulating the baseline l2-switch, with our proposed architecture's IRAP packet loss (Y-axis), simulating the l2-switch extended, for all 25 benchmark videos encoded with 4 different QPs and 10 packet loss rates, in a total of 960 simulations. As one can notice, all points are sub-linear ($y < x$), showing that our approach is never worse than the random one. Instead, the drawn line shows the linear regression with an average IRAP packet loss of 4.8%, 82.5% lower than the average 27.5%. However, even though the linear regression line is below 10%, some points presented worse results.

Fig. 5.2 shows the same data as Fig. 5.1 but colored according to video resolution. It makes clearer that our solution, on average, deals better with lower-resolution videos (416×240 to 1280×720) than with higher-resolution ones (1920×1080 and 2560×1600). Considering the linear regression lines, the 2560×1600 videos show almost 33 times more IRAP packets lost than the 416×240. This happens mainly because of the ratio between the number of packets needed to send an IRAP NAL and the number of packets

Figure 5.1 – IRAP packet loss



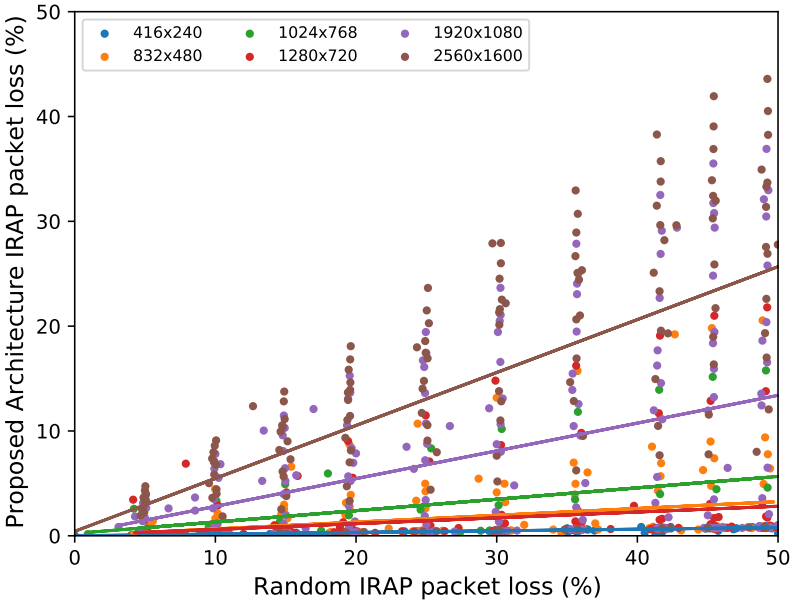
Source: The Author

that the packet-forwarding device can hold in its buffer. Considering that packets will be lost during congestion, higher resolution videos have a higher chance of sending a single frame during the time of congestion, thus decreasing our proposed approach's gain over the baseline. However, results for high-resolution videos vary in the Y-axis, showing a significant margin to improve results.

Fig. 5.3 shows the data points constrained to the higher-resolution videos at 1920×1080 (Fig. 5.4(a)) and 2560×1600 (Fig. 5.4(b)), thus showing the impact of different QPs. As one can notice, the QP affects our proposed architecture when dealing with higher-resolution videos. For the 1920×1080 resolution videos, we can observe IRAP packet loss reduction, on average, in 96% by increasing QP from 22 to 37. For the 2560×1600 resolution videos, we can reduce packets lost, on average, from 21.9% to 3.9% (82% reduction). Overall, our solution provides higher improvements over the random solution for higher QP values. Similar to what happens for high-resolution videos (recall Fig. 5.2), low QP values lead to larger bitstreams and, as a result, to a higher number of packets per IRAP picture. Thus, the ratio of packets to transmit an IRAP picture per number of packets in the buffer grows higher and limits the benefits of our solution.

Combined, these results show that our approach can reduce IRAP packets lost during network congestion at a negligible cost in resource usage and performance degradation. They also show that our proposed architecture works seamlessly for lower-resolution

Figure 5.2 – IRAP packet loss grouped by resolution



Source: The Author

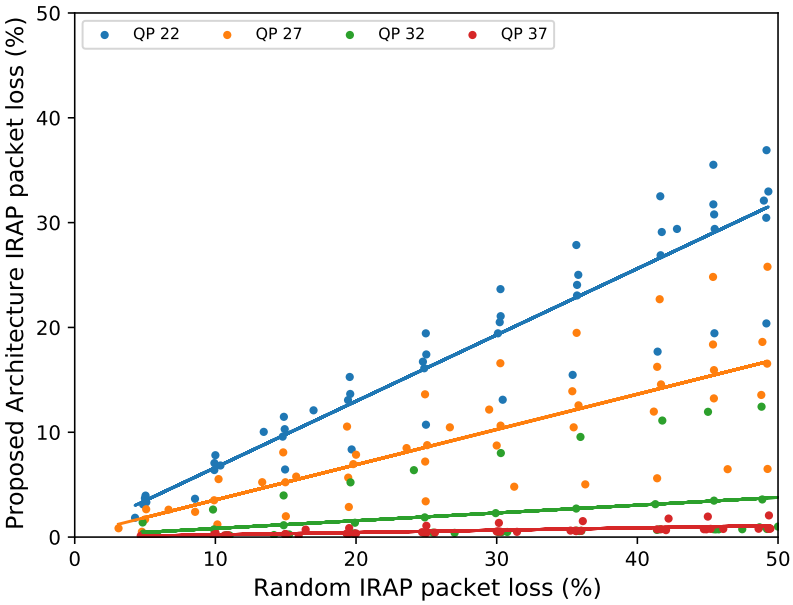
videos but requires attention when dealing with higher-resolution videos, especially when considering the ones encoded with low QPs.

As a second part of our study, we keep using all 25 videos from the CTC of H.265/HEVC (BOSSEN, 2013) encoded using x265 with the four recommended Quantization Parameters (QP - 22, 27, 32, and 37). Now, to encode these videos we use the DecodingRefreshType configuration, which specifies the type of decoding refresh to apply at the intra-frame period picture, as IDR and the Slice Max size, that determines the maximum size a slice can be, according to the Maximum Transmission Unit (MTU). Considering that the videos are relatively short, we chose to concatenate them 100 times before sending them through our simulator, to reach statistical relevance ($\sigma = 0.0068$).

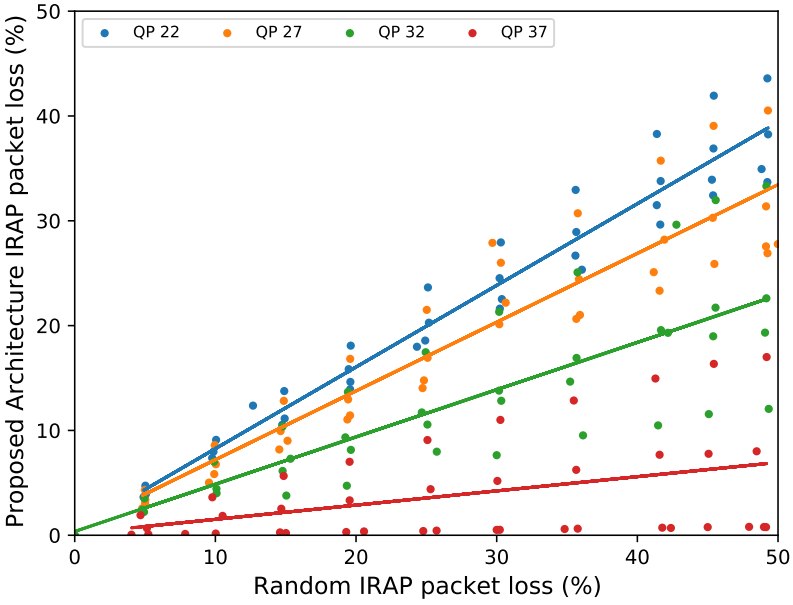
A comparison of the experimental results obtained with the new configurations is presented in Figure 5.4. Upon analyzing this figure and comparing it to Figure 5.1, it can be observed that, in general, the results continue to exhibit a sub-linear behavior. However, the outcomes are comparatively less significant. This is primarily due to the fact that, when adhering to the 1500 byte limit imposed by the MTU, the IRAP frame occupies multiple packets instead of just one. Such behavior results in the ratio of packets with IRAP and non-IRAP being closer to 1, in contrast to the ratio between IRAP and non-IRAP frames that is predetermined by the encoder.

However, as mentioned earlier, simply analyzing the number of lost packets be-

Figure 5.3 – IRAP packet loss grouped by QP.



(a) 1920x1080



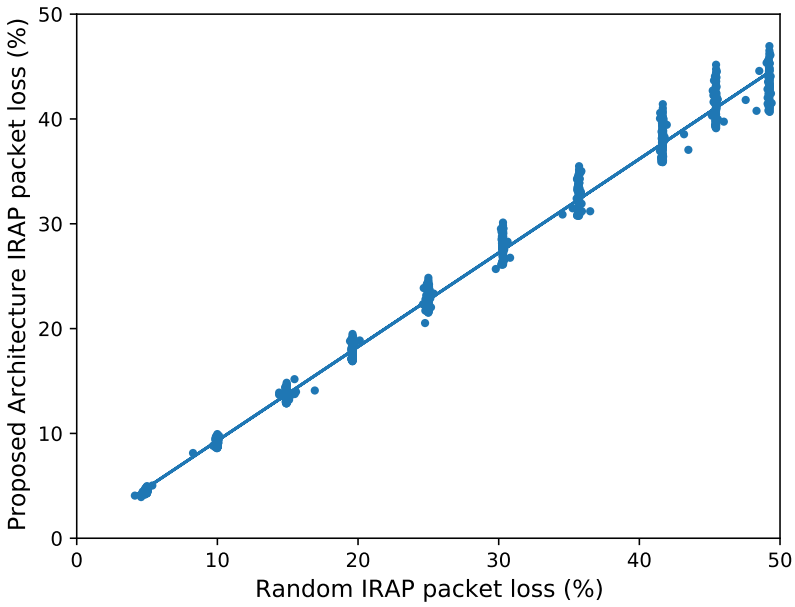
(b) 2560x1600

Source: The Author

tween the two solutions is not sufficient to evaluate whether the final result for the user watching the video is satisfactory. To address this, after conducting simulations, results were obtained during video decoding that indicated the number of frames that could be reconstructed in relation to the original video. Since each video has a different number of

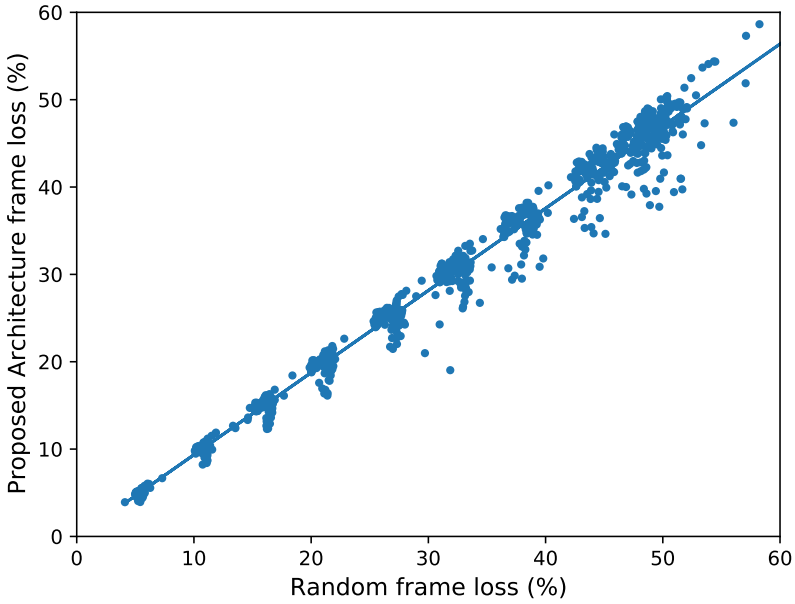
frames, the percentage of frames was used to compare their performance collectively.

Figure 5.4 – IRAP packet loss



Source: The Author

Figure 5.5 – Frame loss Comparison with Random approach



Source: The Author

Fig. 5.5 correlates the random frames loss (X-axis), simulating the baseline l2-switch, with our proposed architecture's frame loss (Y-axis), simulating the l2-switch extended, for all 25 benchmark videos encoded with 4 different QPs and 10 packet loss

rates, in a total of 1000 points (each one representing the our versus random approach). It is possible to see that even considering frame loss instead packet loss our solution continues to win against a random approach, but the results are not too significant as the ones that we got analizing just the packet loss. Our results shows an average of 6.98% more frame loss in the random approach.

In table 5.1 we can see that one of the reasons for these results is that, as the resolution grows, more packets are needed to send just one IRAP frame, so in a congestion situation the 60 slots buffer is not enough to protect those packets. The problem becomes worst in high-resolution videos, where just one IRAP frame from a 2560×1600 video can occupy up to 959 packets at the same time that for a 416×240 resolution video, just 5 packets are enough to send the whole frame. As one can notice the selected QP of the encoding also impacts a lot on the size of the frame itself, because the QP determines the step size for associating the transformed coefficients with a finite set of steps, so as the QP grows, that fidelity of the encoding reduce.

Table 5.1 – Average of packets used by one IRAP frame

Video sequence	Resolution	QP	Packets/IRAP
NebutaFestival	2560x1600	22	959
NebutaFestival	2560x1600	37	179
BasketballDrive	1920x1080	22	108
BasketballDrive	1920x1080	37	15
SlideShow	1280x720	22	62
SlideShow	1280x720	37	31
RaceHorses	416X240	22	25
RaceHorses	416X240	37	5

Source: The Author

This is due to the fact that, considering the way the video is decoded, using IRAP frames as base for other adjacent frames, where adjacent frames, simply put, only store the differences between their frame and the IRAP frame, losing an IRAP frame is a very difficult loss to recover. While losing any other frame decreases the count of decoded frames by 1, losing an IRAP frame can impact the loss of an important sequence of frames.

6 CONCLUSIONS

This work tackled the issue of the ever-increasing stress on the global network infrastructure due to video streaming with the recent technology of in-network computing. We present an algorithm that prioritizes packets containing parts of IRAP frames in a situation of network congestion regardless of the packet forwarding device, a hardware module in the network that can be implemented in FPGA cards, being coupled in parallel to a device, and finally, a modular simulator to evaluate the performance of algorithms in congested environments. Results were considered on two fronts, packet loss, and frame loss. Our comprehensive H.265/HEVC video benchmark results show that our solution can reduce packet loss containing IRAP by over 82% with negligible costs in resource usage and performance. This allocation of resources and performance tends to become less and less impactful when used with more complex network devices, as the experiment was performed on simple layer 2 switch.

Addressing the relationship between packet loss and frame loss in congested networks, with a particular focus on protecting IRAP frames, our findings suggest that just protecting packets containing parts of IRAP frames alone is not sufficient to prevent frame loss in all cases, and more complex algorithms for drop suggestions are needed. This is due to the fact that the video encoding and decoding structure is more complex than just the two classes, IRAP and non-IRAP, of frames, addressed in this study. Additionally, our current solution has demonstrated better performance on lower-resolution video, as higher-resolution video can require up to 200 times more network packets to transmit IRAP frames, making it challenging to maintain them all in the buffer.

As a consequence of this study, we have developed a modular simulation suite for the transmission of video packets under congested conditions. The simulator has several modules that can change everything from video encoding and decoding parameters, how the video is split into network packets, how the simulation of a congested network occurs, to how the packet drop suggestion happens, and what the network equipment does with it. The simulator includes a default configuration included with a default package drop suggestion function, discussed throughout this article. This function can be adjusted in several ways by us and other researchers for further investigations and comparisons between drop suggestion algorithms.

7 FUTURE WORK

Considering that the packet and frame loss results of higher resolution videos were highly impacted by buffer size and although it is assumed that a larger buffer can improve these results, more research is needed to determine its cost-effectiveness. In future work, we intend to improve our approach by considering different drop priorities for non-IRAP frames based on their hierarchy, this hierarchy can be approached as a dependency tree and with that, a package drop policy based on the importance of the type of packets can be defined in this tree. It is important to mention that another approach we intend to take is to explore alternative transmission technologies such as HLS and MPEG-DASH.

A potential avenue needed for future research, aiming to obtain a more accurate simulation of congested networks, would be to simulate multiple network devices and utilize multiple variable data streams to induce congestion on the network in question. This is due to the fact that we cannot actually control the buffer usage of a network device by just passing a stream through it, which leads to competition between all streams passing through the device simultaneously. Therefore, simulating a set of network devices with multiple flows would provide a much more realistic description of congestion in networks than just simulating a device and its buffer.

REFERENCES

BAGCHI, S. et al. Dependability in edge computing. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 63, n. 1, p. 58–66, dec 2019. ISSN 0001-0782. Available from Internet: <https://doi.org/10.1145/3362068>.

BOSSEN, F. Common test conditions and software reference configurations. In: **JCT-VC document no. L1100**. [S.l.: s.n.], 2013.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 3, p. 87–95, jul. 2014. ISSN 0146-4833.

BOSSHART, P. et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In: **ACM SIGCOMM 2013**. New York, NY, USA: ACM, 2013. (SIGCOMM '13), p. 99–110. ISBN 9781450320566.

BOUTEN, N. et al. In-network quality optimization for adaptive video streaming services. **IEEE Transactions on Multimedia**, v. 16, n. 8, p. 2281–2293, 2014.

BRESSANA, P.; ZILBERMAN, N.; SOULÉ, R. A programmable framework for validating data planes. In: **Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos**. [S.l.: s.n.], 2018. p. 1–3.

BRONZINO, F. et al. In-network compute extensions for rate-adaptive content delivery in mobile networks. In: **2014 IEEE 22nd International Conference on Network Protocols**. [S.l.: s.n.], 2014. p. 511–517.

BRUCE, D. What makes a good domain-specific language? apostle, and its approach to parallel discrete event simulation. **Kamin [43]**, p. 17–35, 1997.

CHANDRAKAR, S.; GAITONDE, D.; BAUER, T. Enhancements in ultrascale clb architecture. In: **Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays**. [S.l.: s.n.], 2015. p. 108–116.

CHEN, N. et al. Study on relationship between network video packet loss and video quality. In: **2011 4th International Congress on Image and Signal Processing**. [S.l.: s.n.], 2011. v. 1, p. 282–286.

CHINNERY, D.; KEUTZER, K. **Closing the gap between ASIC & custom: tools and techniques for high-performance ASIC design**. [S.l.]: Springer Science & Business Media, 2002.

Cisco Inc. **Cisco Annual Internet Report (2018–2023) White Paper**. 2021. Available from Internet: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>.

COOKE, R. A.; FAHMY, S. A. Quantifying the latency benefits of near-edge and in-network fpga acceleration. In: **Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking**. [S.l.: s.n.], 2020. p. 7–12.

CORDEIRO, W.; MARQUES, J.; GASPARY, L. Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management. **Journal of Network and Systems Management**, v. 25, n. 4, p. 784–818, Oct 2017. ISSN 1573-7705.

COWAN, C. et al. Adaptive methods for distributed video presentation. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 27, n. 4, p. 580–583, 1995.

DEURSEN, A. V.; KLINT, P.; VISSER, J. Domain-specific languages: An annotated bibliography. **ACM Sigplan Notices**, ACM New York, NY, USA, v. 35, n. 6, p. 26–36, 2000.

DOBRIAN, F. et al. Understanding the impact of video quality on user engagement. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 41, n. 4, p. 362–373, aug 2011. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/2043164.2018478>.

FLOYD, S. Tcp and explicit congestion notification. **ACM SIGCOMM Computer Communication Review**, ACM New York, NY, USA, v. 24, n. 5, p. 8–23, 1994.

FOWLER, M. **Domain-specific languages**. [S.l.]: Pearson Education, 2010.

ITU-T; ISO/IEC. Advanced video coding for generic audiovisual services. **ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 AVC)**, 2003.

ITU-T; ISO/IEC. High Efficiency Video Coding. **ITU-T Recommendation H.265 and ISO/IEC 23008-2**, 2013.

ITU-T; ISO/IEC. **HM reference software for HEVC**. [S.l.], 2023. Available from Internet: <https://vcgit.hhi.fraunhofer.de/jvet/HM/-/tree/master>.

KANG, C.; ALBA, D.; SATARIANO, A. Surging traffic is slowing down our internet. **The New York Times**, March 2020. Available from Internet: <https://www.nytimes.com/2020/03/26/business/coronavirus-internet-traffic-speed.html>.

KAZEMI, M.; IQBAL, R.; SHIRMOHAMMADI, S. Joint intra and multiple description coding for packet loss resilient video transmission. **IEEE Transactions on Multimedia**, v. 20, n. 4, p. 781–795, 2018.

KIEBURTZ, R. B. et al. A software engineering experiment in software component generation. In: IEEE. **Proceedings of IEEE 18th International Conference on Software Engineering**. [S.l.], 1996. p. 542–552.

KORHONEN, J. Study of the subjective visibility of packet loss artifacts in decoded video sequences. **IEEE Transactions on Broadcasting**, v. 64, n. 2, p. 354–366, 2018.

KRUEGER, C. W. Software reuse. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 24, n. 2, p. 131–183, jun. 1992. ISSN 0360-0300.

KUON, I.; ROSE, J. Measuring the gap between fpgas and asics. **IEEE Transactions on computer-aided design of integrated circuits and systems**, IEEE, v. 26, n. 2, p. 203–215, 2007.

KUON, I. et al. Fpga architecture: Survey and challenges. **Foundations and Trends® in Electronic Design Automation**, Now Publishers, Inc., v. 2, n. 2, p. 135–253, 2008.

LADD, D. A.; RAMMING, J. C. Two application languages in software production. In: **USENIX Very High Level Languages Symposium Proceedings**. [S.l.: s.n.], 1994. p. 169–178.

LI, Y. et al. Accelerating distributed reinforcement learning with in-switch computing. In: IEEE. **2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)**. [S.l.], 2019. p. 279–291.

LOCKWOOD, J. W. et al. Netfpga–an open platform for gigabit-rate network switching and routing. In: IEEE. **2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)**. [S.l.], 2007. p. 160–161.

MATSUZONO, K.; ASAEDA, H.; TURLETTI, T. Low latency low loss streaming using in-network coding and caching. In: **IEEE INFOCOM 2017 - IEEE Conference on Computer Communications**. [S.l.: s.n.], 2017. p. 1–9.

NIGHTINGALE, J.; WANG, Q.; GRECOS, C. Hevstream: a framework for streaming and evaluation of high efficiency video coding (hevc) content in loss-prone networks. **IEEE Transactions on Consumer Electronics**, v. 58, n. 2, p. 404–412, 2012.

OZTAS, B. et al. A study on the hevc performance over lossy networks. In: **2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)**. [S.l.: s.n.], 2012. p. 785–788.

RICHARDSON, I. **The H.264 Advanced Video Compression Standard**. [S.l.]: Wiley, 2011.

ROBITZA, W. **h26x-extractor**. 2021. Available from Internet: <https://github.com/slhck/h26x-extractor>.

Sandvine Inc. **The Global Internet Phenomena Report September 2019**. 2019. Available from Internet: <https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/InternetPhenomena/InternetPhenomenaReportQ3201920190910.pdf>.

Sandvine Inc. **The Mobile Internet Phenomena Report 2021**. 2021. Available from Internet: <https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2021/Phenomena/MIPRQ1202120210510.pdf>.

SANVITO, D.; SIRACUSANO, G.; BIFULCO, R. Can the network be the ai accelerator? In: **Proceedings of the 2018 Morning Workshop on In-Network Computing**. [S.l.: s.n.], 2018. p. 20–25.

SAPIO, A. et al. In-network computation is a dumb idea whose time has come. In: **Proceedings of the 16th ACM Workshop on Hot Topics in Networks**. [S.l.: s.n.], 2017. p. 150–156.

SAQUETTI, M. et al. P4vbox: Enabling p4-based switch virtualization. **IEEE Communications Letters**, v. 24, n. 1, p. 146–149, 2020.

SAQUETTI, M. et al. Toward in-network intelligence: Running distributed artificial neural networks in the data plane. **IEEE Communications Letters**, v. 25, n. 11, p. 3551–3555, 2021.

SIRER, E. G.; BERSHAD, B. N. Using production grammars in software testing. **ACM SIGPLAN Notices**, ACM New York, NY, USA, v. 35, n. 1, p. 1–13, 1999.

SIVARAMAN, A. et al. Dc. p4: Programming the forwarding plane of a data-center switch. In: **Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research**. [S.l.: s.n.], 2015. p. 1–8.

SJOBERG, R. et al. Overview of HEVC high-level syntax and reference picture management. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 22, n. 12, p. 1858–1870, 2012.

SUN, P. et al. Optimizing network performance for distributed dnn training on gpu clusters: Imagenet/alexnet training in 1.5 minutes. **arXiv preprint arXiv:1902.06855**, 2019.

SZE, V. **The H.264 Advanced Video Compression Standard**. Switzerland: Springer International Publishing, 2014.

TOKUSASHI, Y. et al. The case for in-network computing on demand. In: **Proceedings of the Fourteenth EuroSys Conference 2019**. New York, NY, USA: Association for Computing Machinery, 2019. (EuroSys '19). ISBN 9781450362818.

TOKUSASHI, Y.; MATSUTANI, H.; ZILBERMAN, N. Lake: the power of in-network computing. In: IEEE. **2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)**. [S.l.], 2018. p. 1–8.

TONI, L.; CHEUNG, G.; FROSSARD, P. In-network view synthesis for interactive multiview video systems. **IEEE Transactions on Multimedia**, v. 18, n. 5, p. 852–864, 2016.

WALLENDAEL, G. V. et al. Keyframe insertion: Enabling low-latency random access and packet loss repair. **Electronics**, Multidisciplinary Digital Publishing Institute, v. 10, n. 6, p. 748, 2021.

WANG, Y. et al. Error resilient video coding techniques. **IEEE Signal Processing Magazine**, v. 17, n. 4, p. 61–82, 2000.

WOODRUFF, J.; RAMANUJAM, M.; ZILBERMAN, N. P4dns: In-network dns. In: IEEE. **2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)**. [S.l.], 2019. p. 1–6.

YANG, H. et al. Review of advanced fpga architectures and technologies. **Journal of Electronics (China)**, Springer, v. 31, n. 5, p. 371–393, 2014.