A Time-Sensitive Network Switch Implementation using P4

Programmable Switches for Real-Time Applications

Prepared by:

John Vusi Mkwebo

MKWJOH004

Department of Electrical Engineering
University of Cape Town

Prepared for:

Dr Joyce Mwangama

Department of Electrical Engineering
University of Cape Town

11 November 2020

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Bachelor of Science degree in Electrical Engineering

Key Words: High-Speed Networking, Programmable Data Planes, SDN, IoT, Industry 4.0

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this final year project report from the work(s) of other people, has been attributed and has been cited and referenced.
3. This final year project report is my own work.
4. I have not allowed, and will not allow anyone to copy my work with the intention of passing it off as their own work or part thereof

Name: **John Vusi Mkwebo**

Signature: **JVM** **Date:** 11 November 2020

Terms of Reference

The terms of reference for this research project is to perform an investigation into the tools required to complete the TSN switch design and implementation. To design and program a switch from scratch requires both understandings of the network protocols that the switch will support and the structure that the switch should resemble. In addition, knowledge of the p4 language construct is required to make the mapping from switch design to implementation.

Once the investigation has been carried out and the problem is identified, formulated, analysed, and solved, a TSN switch will be implemented. Once the switch is implemented, the design of the TSN network and the appropriate controller is to be completed. Experimentations will be performed to determine whether the TSN operational specifications are achieved.

Acknowledgements

My sincere appreciation goes to my supervisor, Dr J Mwangama for her support and cooperation that enabled me to conduct this research project smoothly despite the difficulties faced due to COVID-19 pandemic.

Sincere gratitude to my girlfriend Yonela Mdledle, thank you for all the emotional support and always being a helping hand during this roller coaster journey. Words cannot even describe my gratitude.

To my friends Francis Mogale, Thabo Yiga, and many more, thank you for all the financial, academic support and always being there any way possible.

Special thanks to Faculty of Engineering & the Built Environment (EBE) communications and marketing manager, Mary Hilton and the unknown funders for food and accommodation assistance.

To Jess Smith, thank for you for your literature expertise and good work in editing my literature review.

Abstract

Till this day, everyone is used to the common models of internet protocols such Local Area Network (LAN) and the internet. However, these models operate based on best-effort mode and all their metrics based on average delay, speed, bandwidth, etc. These are great networks, and we use them every day but are not suitable applications that require high or deterministic network availability.
Time-Sensitive Networking (TSN) is set of evolving IEEE standards designed to allow Ethernet based LANs to be able to provide services to real-time/time-sensitive applications traffic in addition to best-effort/background traffic. TSN started as audio/video bridging (AVB) which was an IEEE 802.1BA standard created in 2011 and it was developed and renamed to this new set of technology called IEEE 802.1Q. The main takeaway of TSN is that it allows both time-sensitive traffic and best-effort traffic to coexist on the same network, with time-sensitive traffic being deterministic and prioritized over best-effort traffic. TSN is getting an attention from real-time application environment such as Industrial Internet of Thing (IIoT), Pro A/V, and Automotive systems.

The paper will explore Time-Sensitive Networking using v1model programmable virtual switch. The v1model switch is P4 architectural model switch that supports implementation of network data plane programming and forwarding techniques using Programming Protocol-Independent Packet Processors (P4). P4 is a high-level networking language framework that describes what network function should be executed and how the overall network data plane processing pipeline should be organised. A TSN switch and network is implemented on Mininet making a use Software Defined network to manage communication of the network.

This research project consists of a literature review, which introduces the state of the art of the topic, followed by design requirements, system design and implementation, result and discussions, and finally, conclusion and recommendations. The results show that real-time/time-sensitive traffic can be given priority over best-effort traffic to guarantee the requirement of best Quality of Service (QoS).

List of Figures

List of Tables

List of Listings

List of Abbreviations

TSN	Time-Sensitive Networking
SDN	Software-Defined Networking
P4	Programming Protocol-Independent Packet Processors
IoT	Internet of Things
IIoT	Industrial Internet of Things
LAN	Local Area Network
CAN	Controller Area Network
QoS	Quality-of-Service
VXLAN	Virtual Extensible Local Area Network
IoT	Internet of Things
PARC	Palo Alto Research Centre
JSON	JavaScript Object Notation
MAC	Media Access Control
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
M2M	machine to machine
OSI	Open System Interconnection
gPTP	generic Precision Time Protocol
BMCA	Best Master Clock Algorithm
GCL	Gate Controlled List
GCEs	Gate Control Entries
CNC	Centralized Network Configuration
CUC	Centralized User Configuration
UNI	User-Network Interface
API	Application Programming Interface
TSSDN	Time-Sensitive Software-Defined Networking
VLAN	Virtual Local Area Network
PISA	Protocol Independent Switch Architecture
ALU	Arithmetic Logic Unit
DAG	Database Availability Group
IPv4	Internet Protocol version 4
RPC	Remote Procedure Call
CPU	Central Processing Unit
TSMP	Time-Sensitive Management Protocol
CRC	Cyclic Redundancy Check
TTL	Time To Live
BMv2	Behavioral Model version 2
CLI	Command-Line Interface
SCTP	Stream Control Transmission Protocol
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
RTT	Round-Trip Time
FPGA	Field-Programmable gate Array
ASIC	Application-Specific Integrated Circuit
ICS	Industrial Control System
PLC	Programmable Logic system
WAN	Wide Area Network
AVB	Audio/Video Bridging

Table of Contents

1. Introduction

1.1 Background to the study

Looking at the history of industrial standards, communication systems made use of real-time protocols such as Controller Area Network (CAN), Profibus and Modbus which form part of IEEE 802.3 standard. The growth in IoT and Industry 4.0 allowed high-speed industrial applications which introduced a rapid growth and improvement towards Ethernet real-time applications. IEEE 802.3 standard is however non-deterministic to real time applications due to its lack of the ability to respond in a time-bounded manner [1]. These challenges have been addressed by the IEEE 802.1 Working Group in developing TSN which enables a deterministic communication on the Ethernet network.

Time-Sensitive Networking (TSN) is an Ethernet extension proposed by IEEE and it is designed to make Ethernet-based networks with multiple communication devices like factories, automotive and audio and video performance to benefit from real-time communication from TSN standards [7]. In recent years, Ethernet appears to have become the attention for industrial and automotive applications. The IEEE802.1 standard such as Traffic Scheduling (IEEE802.1Qbv & IEEE 802.1Qbu), Time Synchronization(IEEE802.1AS) and System Configuration (IEEE 802.1Qat & IEEE 802.1Qcc) for TSN have proven to have conquered the real-time determinism and robustness requirement of these environments [2]. Furthermore, Software-Defined Networking (SDN) model has changed data centre networks by separating the control plane from data plane of network devices to a central unit with a global network knowledge that ensures simple packet fast-forwarding at devices [2]. However, SDN does not ensure Quality-of-Service (QoS) to the forwarding planes. Programmable TSN devices present potential value to the flexibility required in real-time application environment.

The research will implement programmable TSN switches using Programming Protocol-independent Packet Processors (P4). Although SDN decoupled control and data plane specifies the API between these planes, it still assumes that the behaviour of the data plane to be fixed which does not give the greatest amount of flexibility to network owners who remain dependent on equipment manufacturers. For example, deploying a new network protocol such as VXLAN can take up to 4 years between the initial proposal and its commercial availability in high-speed devices, which is clearly an obstruction to innovation [15]. Industry and academia developed P4 to be a new domain-specific programming language specification open to the public to overcome existing challenges.

1.2 Objectives of this study

1.2.1 Problems to be investigated

The objective of this study is to conduct a literature review on Time-Sensitive Networking, Software Defined Networking (SDN), Programming Protocol-Independent Packet Processors (P4) and Mininet emulation environment. Analyse the findings and formulate them to implement a TSN switch and deploy the switch on a designed TSN network incorporating an appropriate controller.

1.2.2 Purpose of the study

The purpose of this study is show how TSN specifications to support real-time applications can be achieved using programmable switches, P4 and SDN.

1.3 Scope and Limitations

TSN targets different segments and can be used in different markets and each market has different sets of requirements. This research project is limited to implementation of TSN switch and deploy the switch on a TSN network consisting of various switches and a network controller. The network should be able to achieve TSN operational specifications, by giving one traffic class priority over another coexisting different traffic class on the same network. This study does not show how different factors such as, how network size influences the behaviour of the network.

1.4 Plan of development

This section outlines the structure of the research project and provide a brief description of the contents of each chapter. The paper is organised as follows:

- Chapter 1 contains an introduction of the paper and the layout is made up of background, objectives, scope and limitations, and plan of development of this paper.
- In chapter 2 is a literature review, this is a state of the art of the project topic and highlights the concept of Ethernet, Time-Sensitive Networking, Programming Protocol-Independent Packet Processors, Software Defined Networking, and related studies.
- The system design requirements are covered in chapter 3, this chapter provides the requirements to be performed by the system to fulfil the TSN specifications. It further provides the functions to be performed to meet the requirements and draws the acceptance test protocol to be met.
- Chapter 4 contains the system design and implementation methods carried out to design, implement the TSN switch and deploy various switches on TSN network consisting of a controller. Here all the system architecture and valuable code listings are provided.
- Chapter 5 provides the results obtained from experiments and are discussed. It also covers the test setup environment.
- Chapter 6 conducts conclusions.
- Chapters 7 provides recommendations for possible improvements and future work.

2. Literature Review

To understand the research topic, the first step was to conduct a literature review. This section covers and summarises the state-of-the-art works related to this topic conducted by prior researchers. This section will cover an overview of Time-sensitive Networking and Switched Ethernet and then shifts focus to reviews related to this research project.

2.1 Ethernet

Ethernet is a group of networking technologies for communication over LAN. It was developed by XEROX Palo Alto Research Centre (PARC) and it is an IEEE 802.3 Standard. Ethernet is in the second layer of OSI model which is the lowest and physical layer. Topologies like bus, tree, line, and ring are supported by Ethernet [3]. Over the years, manufacturing systems and devices such as sensors, actuators, and controllers have developed to have real-time requirements which are supposed to run on the Ethernet. However, one impediment of Ethernet is its nondeterministic behaviour that cannot satisfy the real-time requirements. It is for this reason that switched Ethernet was developed to become an alternative to solve industrial real-time systems requirements and to get rid of Ethernet uncertainties.

2.1.1 Switched Ethernet

Switched Ethernet make use of high bandwidth, cost efficiency, and scalability of Ethernet and delivers full-duplex links, temporal reliability, bounded and low jitter. Figure 2.1 shows two network topologies to demonstrate the difference between conventional Ethernet which uses a hub to forward messages and a modern switched Ethernet which uses a switch. A hub is a central node that upon arrival of message, it transmits the message to all other nodes in the network, whereas a switch has the capability to determine the address of the received message and only forward it to the destined node in the network. A switch achieves this functionality by creating a MAC address table used to store MAC addresses of all connected notes nodes and uses it to match received messages, this allows multiple messages to be transmitted in the network simultaneously for different receivers. A switch is also able to transmit and receive messages simultaneously and is said to provide a full-duplex link and hub a half-duplex link. These characteristics of switched Ethernet allow it to support real-time traffic solutions such as AVB and TSN [3].

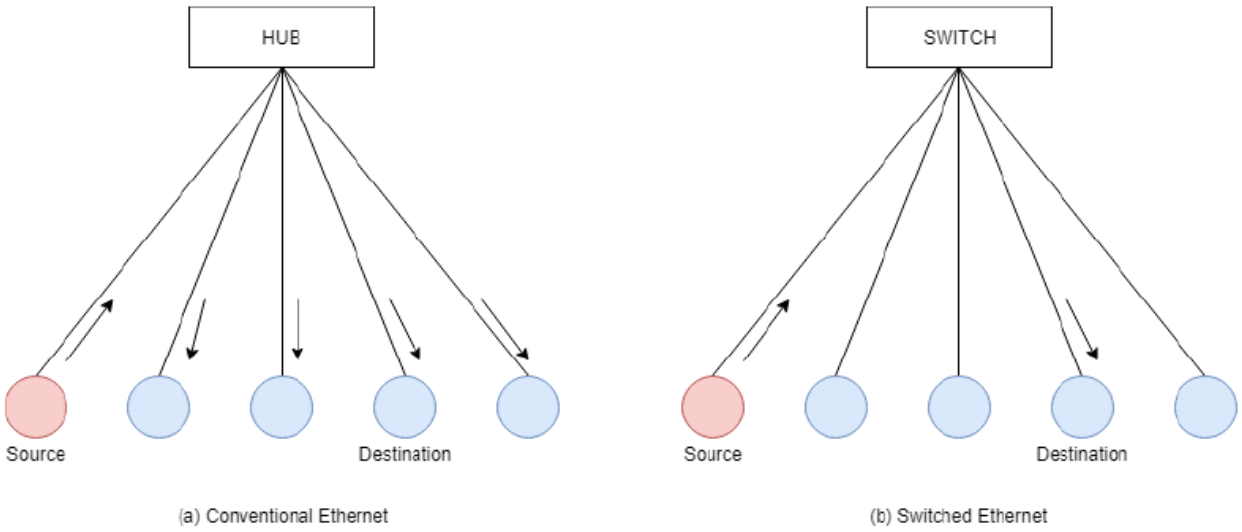


Figure 2.1: Conventional & Switched Ethernet topology [3]

2.1.2 Real-Time Ethernet

Thomas Brand [33] highlights that guaranteed latency and corresponding determinism are no longer achieved by classic Ethernet due to a rising demand for secure data transmission and real-time requirement. New applications that comes with IoT and Industry 4.0 present new challenges with technologies that require better bandwidth allocation to be controlled faster and correctly, which causes shared Ethernet to be inaccurate for use in the industrial environment, automation and process control [3]. This led to the development of real-time Ethernet solutions such as PROFINET*, EtherCAT*, Sercos*, EtherNet/IP*, CCLink-IE*, Modbus TCP* and Ethernet PowerLink* [1], however, the industrial environment is equipped with hardware that supports typical TCP, UDP, IP protocols which do not offer real-time characteristics. This is where Time-Sensitive Networking (TSN) comes in. TSN a is combination of IEEE 802.1 standards, which aims at uniformity of all industrial communications by combining conventional Ethernet with real-time functionality required for industrial communications [33]. This setting is shown in Figure 2.2, depicted by protocol A and protocol B. These TSN capabilities ensures real-time determinism and allows it to operate up to 10 Gbps [33]. The functionalities of TSN have resulted in growing research and development by different entities, particularly in the industrial and automation environment.

2.2 Time-Sensitive Networking

Time-Sensitive Networking (TSN) is an evolving set of standards developed by IEEE 802.1 Task Group to allow time-sensitive traffic and best effort traffic to co-exist on the same Ethernet network. It started as Audio/Video Bridging (AVB) and developed to improve AVB and renamed to TSN. TSN technology mainly focuses on four ethernet aspects: synchronization between communicating devices, end-to-end bounded latency, allows real-time traffic stream reliability, and management of network resources [21].

The capabilities of the TSN lies in layer 2, which is the data layer in the Open System Interconnection (OSI) model as indicated in Figure 2.2. TSN IP encapsulation functionality, data link layers improves the operation to manage transmission errors, flow data regulation and to improve a well-defined industrial protocol such as OPC Unified Architecture which is a form of machine to machine (M2M) protocol for industrial automation [1].

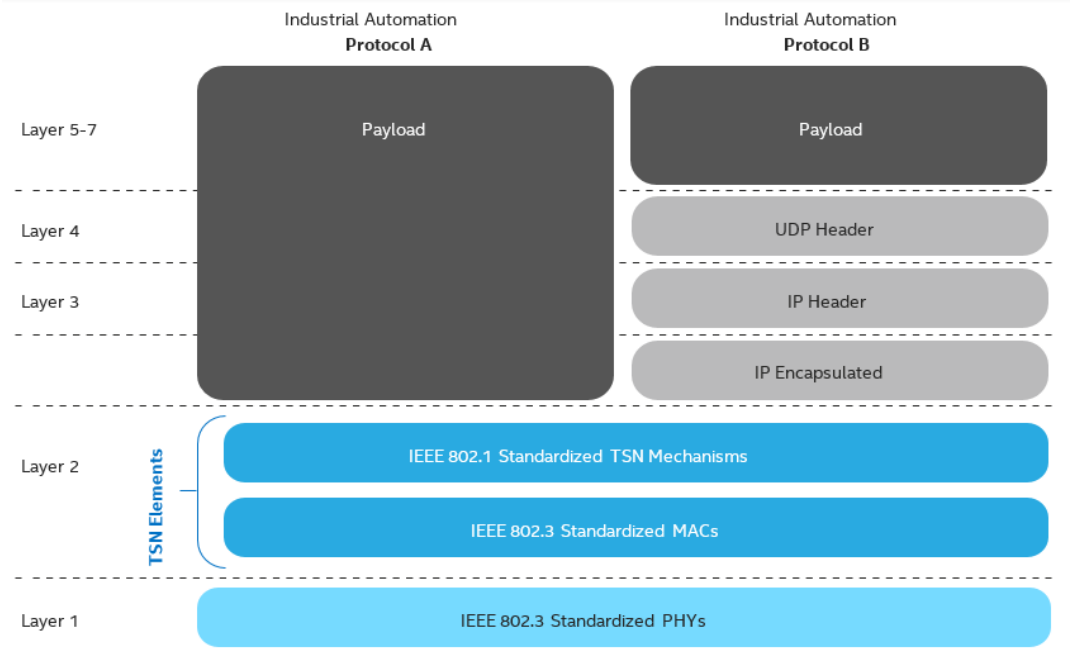


Figure 2.2: OSI Layers [1]

TSN belongs to IEEE 802.1 which is split into different standard sets of networking areas, and each standard performs based on the systems data priority. These standards address topics such as timing, synchronization, forwarding, queuing, seamless redundancy, and stream reservation to extend the functionality of Quality of Service (QoS) and guarantees data transmission through switched networks for industrial communication protocols [14]. Figure 2.3 shows TSN IEEE 802.1 standards and the features supported by each standard.

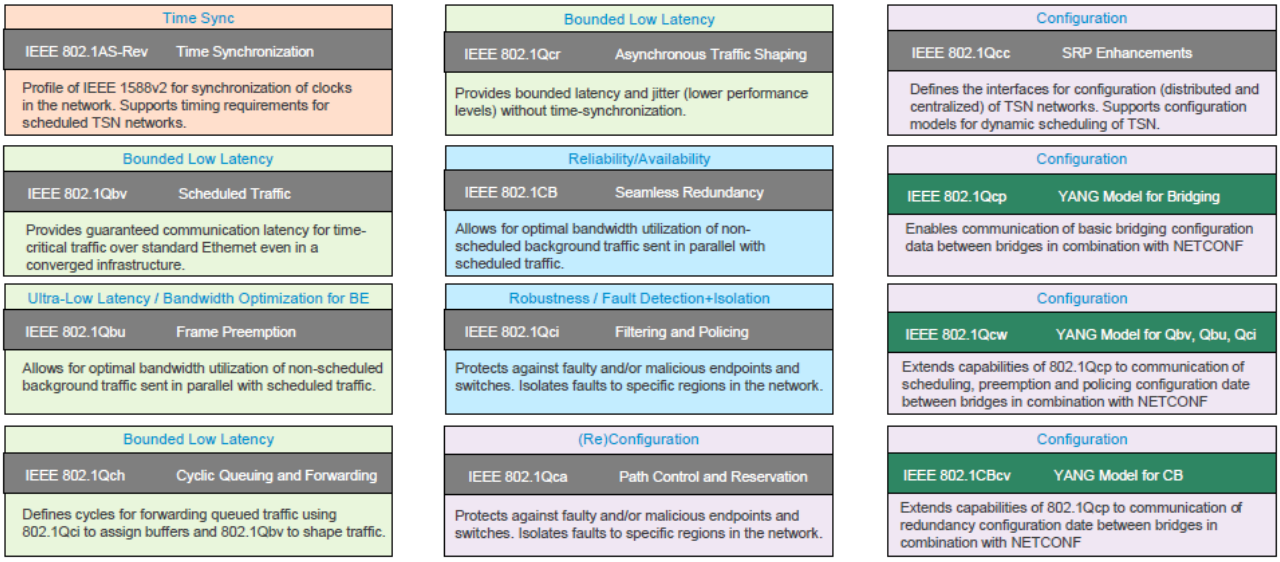
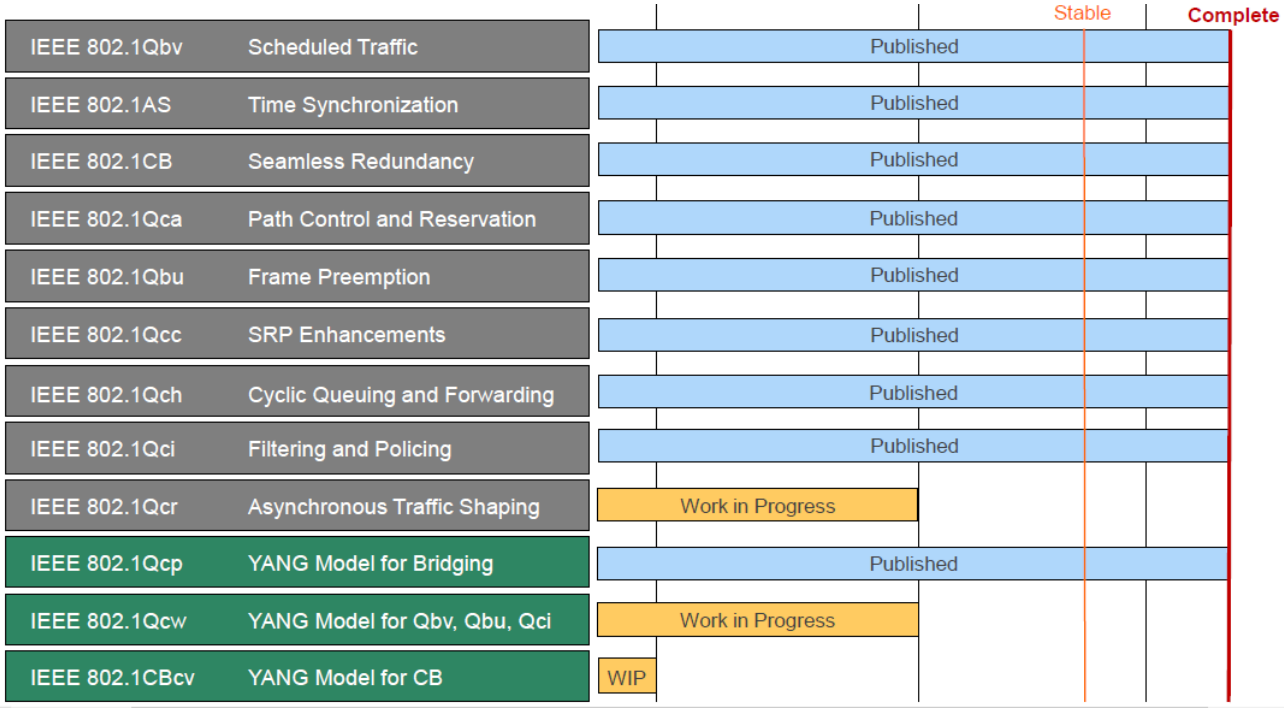


Figure 2.3: TSN IEEE 802.1 standards [23]

Furthermore, Figure 2.4 shows TSN standardization complete status in 2015 [33], and subsequently the complete standards and ongoing going standards in the year 2019 [23]. This shows how rapidly research and development of TSN standards have grown in just four years.



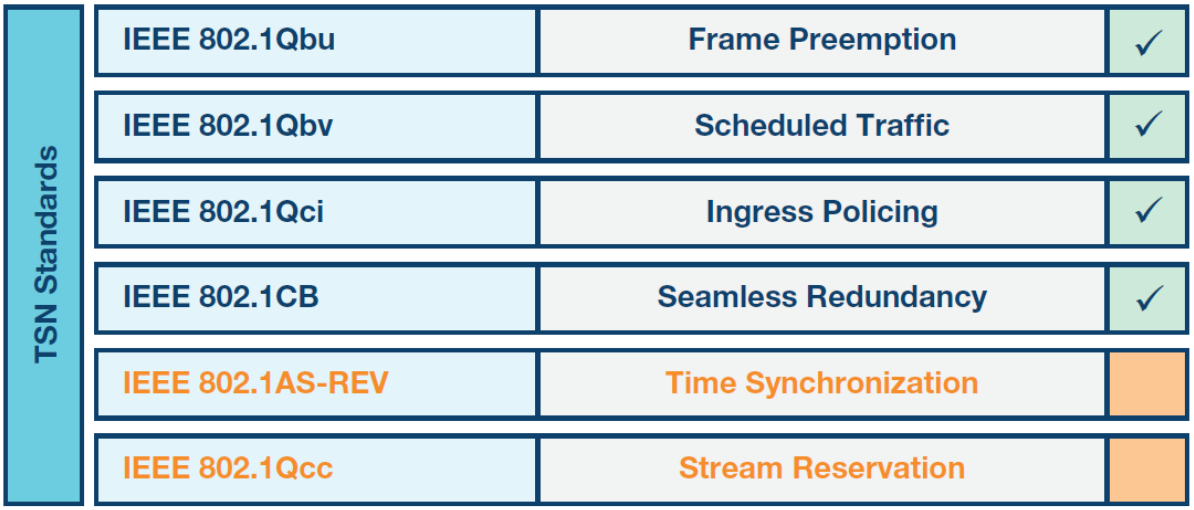


Figure 2.4: 2019,2015 TSN standardization status [23,33]

This research project will only use features which are said to be the heart of TSN mechanisms [23], due to their ability to provide time synchronization for devices and scheduled forwarding of traffic flow through the network to deliver deterministic communication over standard Ethernet. These standards are, IEEE 802.1AS (Time Synchronization), IEEE 802.1Qbv (Traffic Scheduling), IEEE 802.1Qav (Traffic Shaping), IEEE 802.1Qcc (Systems Configuration) and IEEE 802.1Qat (Stream Reservation). Further explanation of each of the five standards is conducted on the subsequent subsections.

2.2.1 IEEE 802.1AS Time Synchronization

Most of the TSN standards are aimed at establishing a common time reference that is shared across TSN network devices [30]. Time synchronization is used to determine data and signal scheduling control and is achieved by using the IEEE 802.1AS standard adopted from IEEE 1588-2008 and also known as the generic Precision Time Protocol (gPTP), which uses a special outline to synchronize clocks between network devices [30]. Figure 5 shows a time-aware network used in IEEE 802.1AS to synchronize time by passing real-time messages between the clock master and slaves.

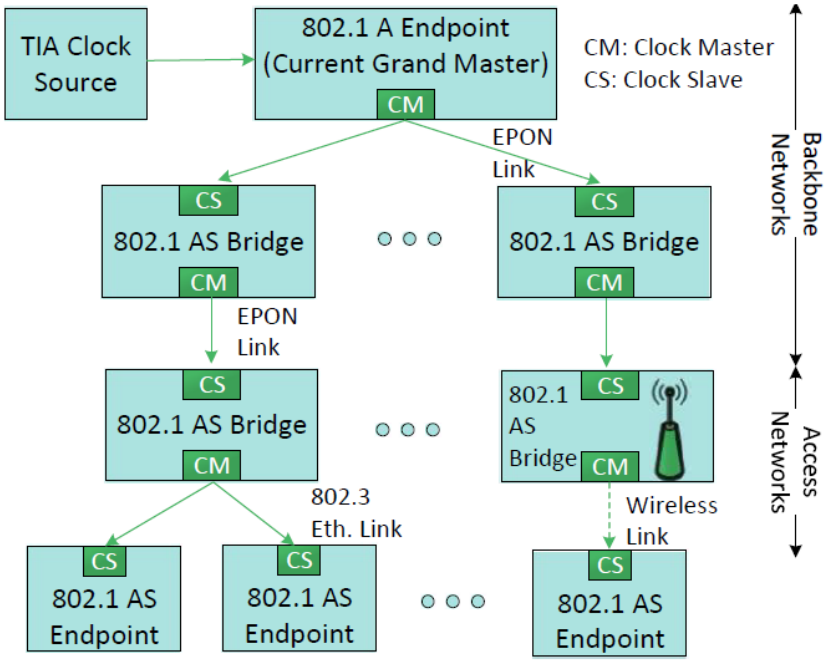


Figure 2.5: Time-Aware Network [30]

As shown in the figure, time synchronization occurs in two stage process. Firstly, the gPTP devices will establish a master-slave hierarchy, followed by clock synchronization operations application [30]. The master-slave hierarchy is split into two algorithms to compare data and state decision, this establishment is known as the Best Master Clock Algorithm (BMCA).

2.2.2 IEEE 802.1Qbv Traffic Scheduling

The IEEE 802.1Qbv controls queues of the egress port to be transmitted using a programmed gating mechanism that opens/closes according to a known and agreed upon time schedule. The transmissions are controlled by the Gate Controlled List (GCL) with multiple Gate Control Entries (GCEs) that determines which queues are open [30]. A model of IEEE 802.1Qbv performing traffic scheduling is shown in figure 2.6.

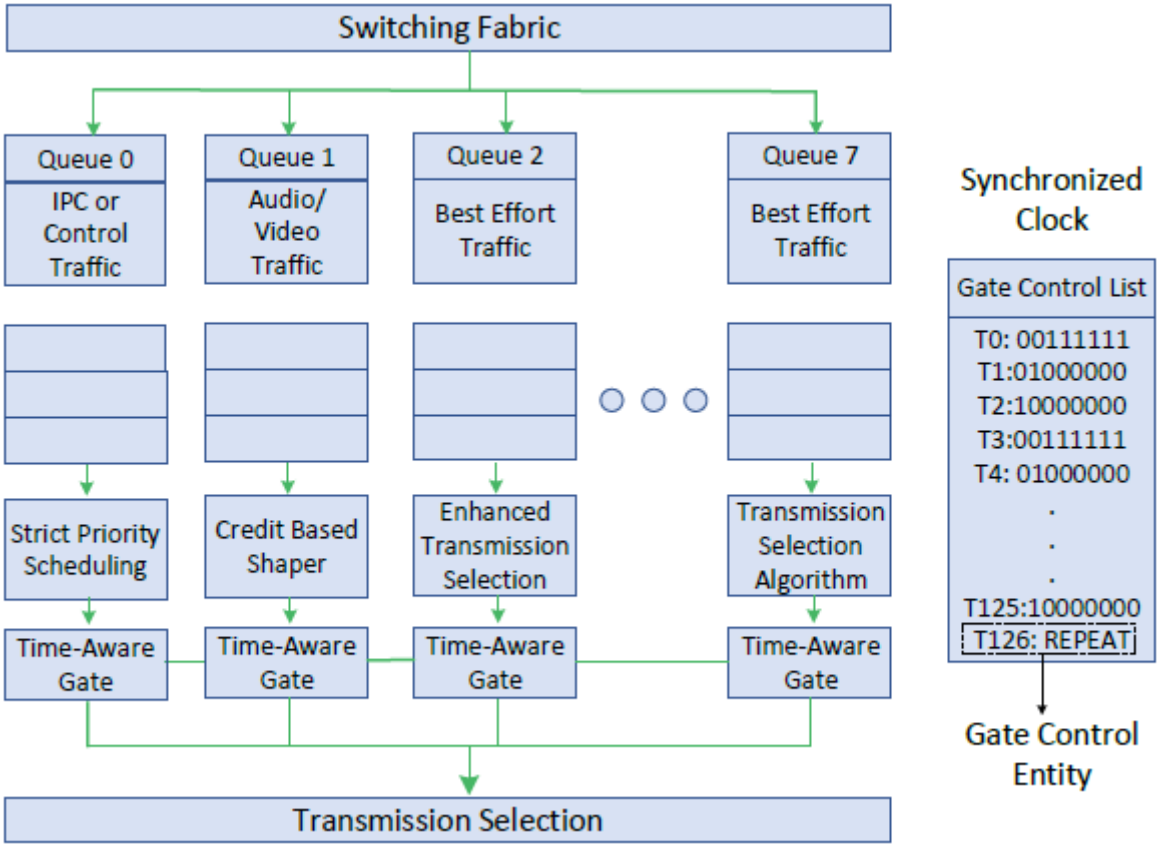


Figure 2.6: IEEE 802.1Qbv [30]

The scheduling or the gating mechanism is achieved by programming the sequence of the gate events consisting of relative time-stamp(ti) to the previous event in the sequence and a bit-mask showing queues which are to be considered for transmission selection until the next event [27]. The process is continued for pre-programmed duration of Tcycle, after which the gate events will have a cyclic schedule of length Tcycle. Moreover, in incorporating IEEE 802.1AS to this mechanism, all clocks of the switches in the network can be synchronized, aligning all cyclic gate events in the network.

2.2.3 IEEE 802.1Qav Traffic Shaping

Traffic shaping is implemented on TSN/AVB switches to avoid extensive packet queueing and to ensure that maximum latency between sources and sinks is not exceeded. The function of this topology is to manage the pricing of frames into queues on the ingress port and select these frames based on their priority for transmission. The priority of frames is also referred to as traffic classes, and

these classes can be categorised as time-sensitive or time-critical with strict priority traffic, time-sensitive without strict priority and non-time-sensitive traffic. The switch determines the different classes by reading the three priority bits of the VLAN Tag of the incoming frames and matches them to the table that maps the priorities to traffic classes or queues. The selection of different traffic classes can be achieved using selection algorithms such as Strict Priority and Credit Based Shaping.

2.2.4 IEEE 802.1Qat Stream Reservation

The stream reservation is a network management reservation protocol for traffic streams such as real-time applications that require guaranteed Quality of Service (QoS) in Local Area Networks (LANs). The main task of the protocol is handled within Multiple Stream Registration Protocol (MSRP) and Multiple Registration Protocol, these protocols are to transverse the attributes through the LAN to allow traffic streams to register communication initialization between talkers and listeners, this information is propagated through the network. This protocol is shown in Figure 2.7 below.

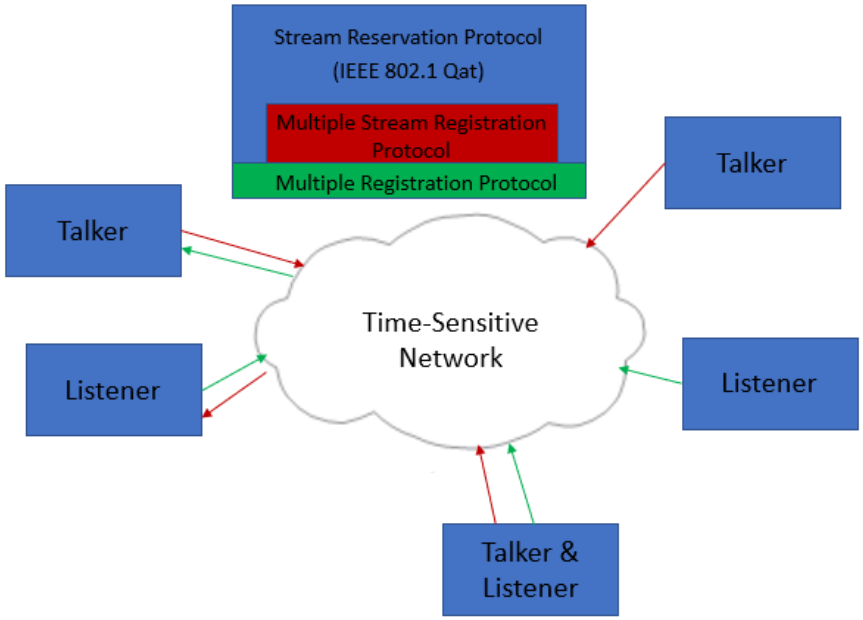


Figure 2.7: Stream Reservation Protocol

The talkers in the network send messages indicated in red arrows called talker MSRP message, the purpose of the messages is to find any listeners in the network willing to exchange communication, any listener available for communication will respond with an MSRP message ready, this is indicated in green arrows. Once the MSRP message "I am ready" from a listener arrives to the talker, the TSN network consisting of TSN switches will reserve resources for traffic exchange between the communicating hosts to ensure deterministic quality of service.

2.2.5 IEEE 802.1Qcc Systems Configuration

Lastly, the TSN IEEE 802.1 standard to be examined for this project is the IEEE 802.1Qcc. This an enhancement of the Stream Reservation Protocol (802.1Qat) by adding more support to account for more streams, has a capability of configurable reservation classes and streams, support layer 3 streaming and user network interface for routing and reservation [14]. The standard can have a three different architectural model realization, and the models are: fully distributed, centralized network and fully centralized as shown in figure 2.8. However, this project utilizes the centralized Network configuration model, which does not make use CUC.

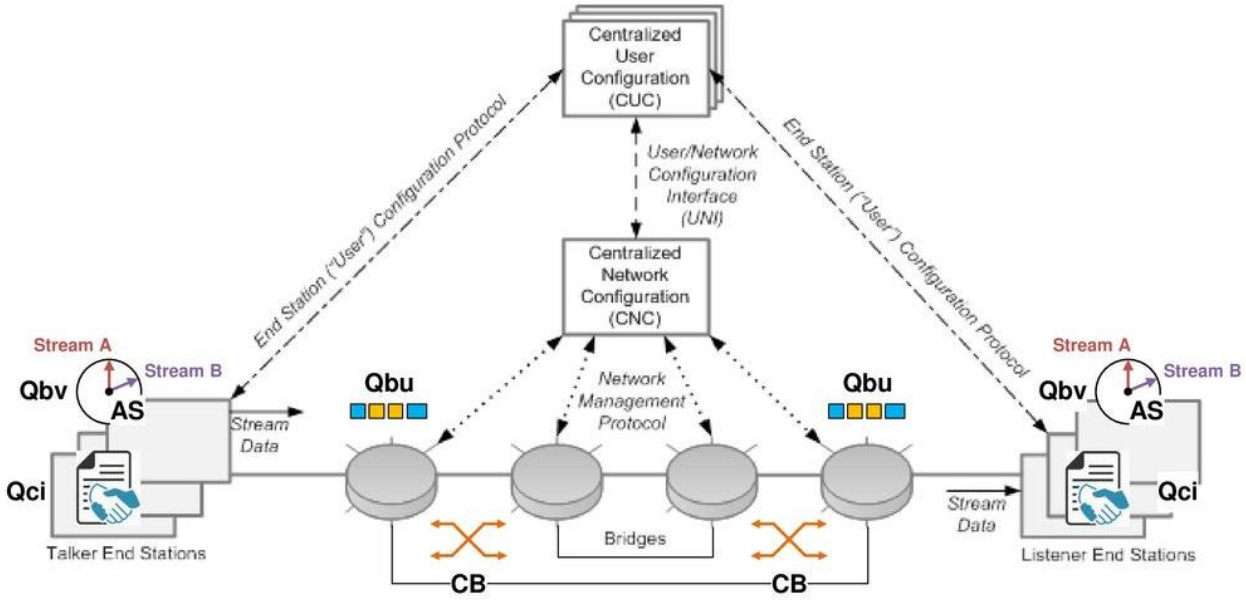


Figure 2.8: IEEE 802.1Qcc -Fully centralized model [31]

The CNC model has a global knowledge of all the devices and streams in the network. The end point in this model would still use UNI protocol to send stream requests, however, the first bridge will act as a pipeline to direct the request to the CNC which will configure the bridge after finishing the first computation and for generational directions [31].

2.3 Programmable Real-Time Ethernet Communication

The little work in combing the concepts of TSN and SDN and lack of simulation environment that supports programmable real-time networks gave Häckel et al [2] an opportunity to introduce programming options for real-time Ethernet devices to improve flexibility, adaptability to the changing industrial environment, and security. This idea plans to factor in all three layers of the SDN concept, namely; Forwarding devices must provide a programming interface using open standards, control functionality for real-time must be extracted from the switches and integrated into SDN controller, and controller applications must be able to program and manage real-time devices [2].

2.3.1 Switch Architecture

The concept of real-time Ethernet and SDN to achieve TSN application requires switches to be supplanted with modules to increase the operation of normal switching hardware, is shown in Figure 2.8. Ingress control module is a module that has been added to the real-time communication Ethernet. This module is represented as the “Per-Stream Filtering and Policing” in TSN and is used as a filtering tool for incoming Ethernet frames and for managing arrival times and bandwidth [2]. The same approach is implemented at the ingress of a switch to apply a priority queuing, real-time scheduling, and traffic shaping. This is one of the components of TSN and key IEEE 802.1 standard known as “Enhancements for Scheduled Traffic” which is an IEEE802.1Qbv specification. This traffic scheduling allows for different traffic classes to coexist with competing priorities on the same network [1] and uses a Gate Control List (GCL) to indicate which priorities are forwarded to which port at what specific time.

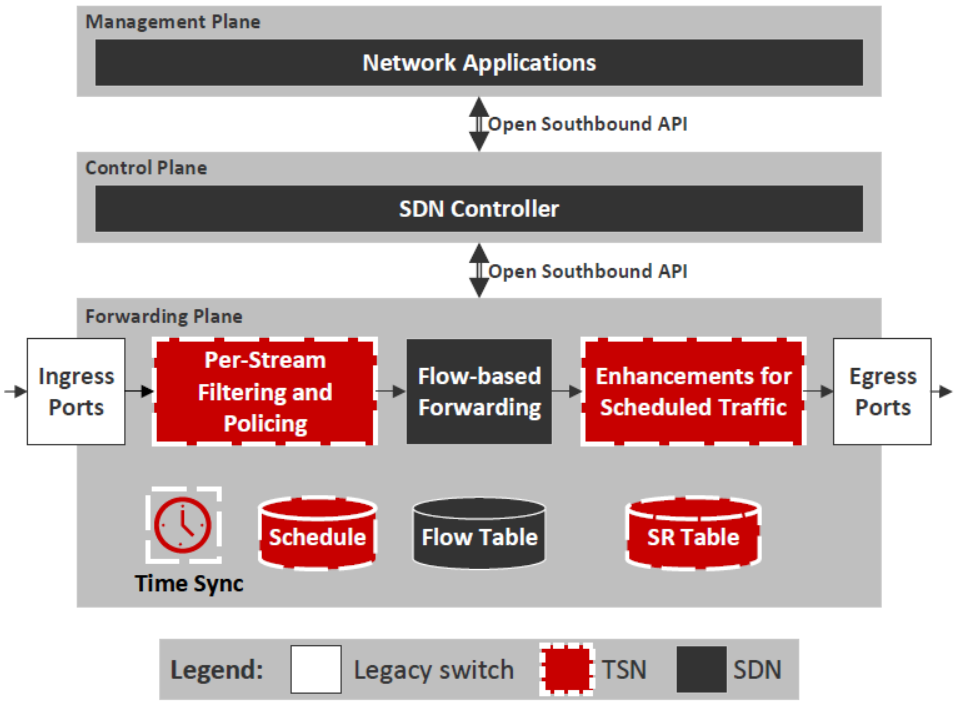


Figure 2.9: Time-Sensitive SDN real-time switch [7]

The scheduling information is stored in the "schedule" tables and is synchronized across all communicating devices in the network, which is controlled by "Time Sync" module. "SR Time" contains all talkers and listeners for time-sensitive applications [2].
In an SDN switch, the forwarding module of a standard Ethernet switch is replaced by a flow-based forwarding module that performs flow table lookups based on packet match rules [2]. The SDN controller executes MAC-Address learning, topology discovery and route selection. The southbound API programming interface between the SDN controller and switch implements OpenFlow protocols. The northbound API is responsible for other network applications.
In programmable real-time network, real-time and SDN switching are merged to achieve flow-based functions. The ingress and egress control modules must stay unaltered by additional programming to ensure that real-time functions are not altered [7].

2.3.2 Implementing Time-Sensitive Flows with OpenFlow

An extension for Time-Sensitive Software-Defined Networking (TSSDN) match rules to OpenFlow is required to manage the forwarding of TSN flows [7]. The supplanted control information of the SRP needs to be handled between the TSSDN switches and the controller. Forwarding in a TSN switch is carried out based on the destination MAC address which identifies the stream's sinks group [7]. The use of SDN presents an opportunity to perform forwarding with multiple additional match fields that enables non-functional requirements realization. Table 2.1 shows how match field and OpenFlow can be used together to forward time-sensitive flows.

	Match field	OpenFlow identifier
Listener Group	Eth. Dst. Addr.	OFPXMT OFB ETH DST
Talker Address	Eth. Src. Addr.	OFPXMT OFB ETH SRC
Ingress Port	Switch Ingress Port	OFPXMT OFB IN PORT
VLAN ID	802.1Q VLAN ID	OFPXMT OFB VLAN VID
Stream Priority	802.1Q Priority	OFPXMT OFB VLAN PCP

Table 2.1: OpenFlow match used for TSN streams [7]

Matching the source of the MAC address of the talker in the network ensures the stream always originates from the same talker, this avoids the misuse of the MAC addresses. How this works: if the TSN path redundancy feature is not in use, one of the streams should always arrive at the same ingress of the switch. However, if it does not, SDN controller is then informed of the change in the network. VLAN ID and VLAN Stream priority MAC addresses are used as multiple and with their streams in streams in place, TSN flows can be identified and forwarded correctly [7].

The paper by Häckel et al [2], investigates switching that combines TSN and SDN. This paper uses simulation model in OMNET++ and this SDN does alter or improve the real-time capabilities, only using OpenFlow to map TSN streams on SDN controller. However, this project utilises Mininet which is a virtual emulation in real-time and adding more parts programmable switching of TSN in SDN with a possibility to improve real-time capabilities.

2.4 P4: Programming Protocol-Independent Packet Processors

Software-Defined Networking (SDN) has given owners and operators control over their network control plane [34]. SDN allows the control plane to be physically separated from the forwarding plane, and a single plane can control multiple forwarding devices. However, SDN does not give flexibility to owners and operators who remain reliant on equipment manufacturers. It is due to this issue that led to programmable data planes using languages such as P4.

P4 is a language written to configure how packet should be processed on the switch. Figure 2.9 shows the relationship between P4 language configuration, SDN control plane, OpenFlow, and target switch.

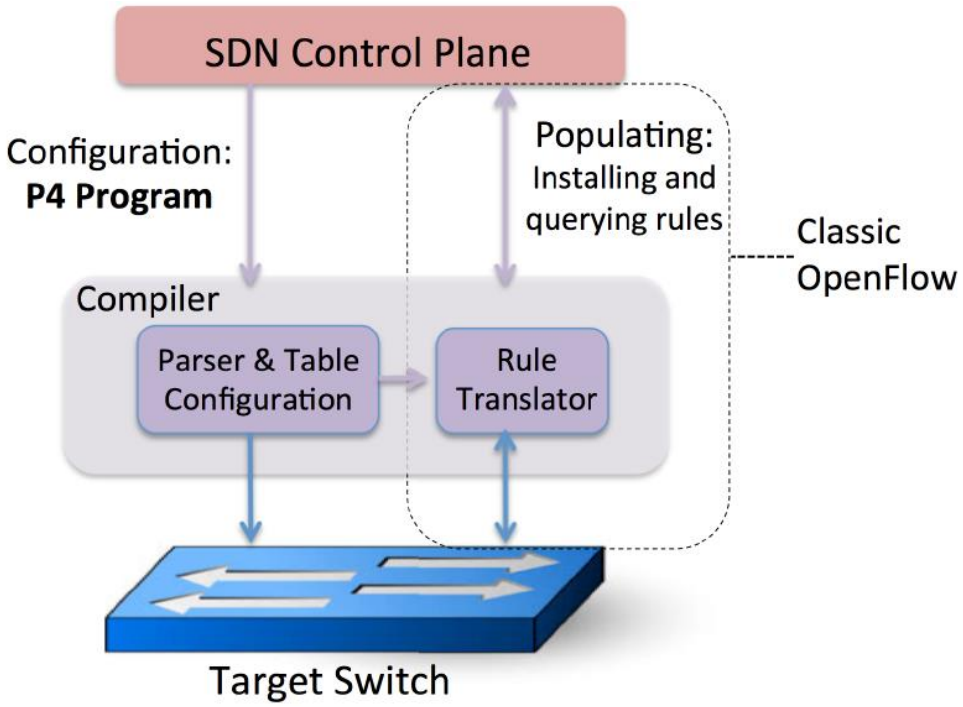


Figure 2.10: Flow of P4 Configuration [10]

As shown in Figure 14, the compiler will compile the P4 language down to run on the targeted switch. However, instead of a fixed-function switch, this will run on a class machine such as Protocol Independent Switch Architecture (PISA) [34]. Figure 2.10 below shows a programmable switch chip pipeline and control flow graph.

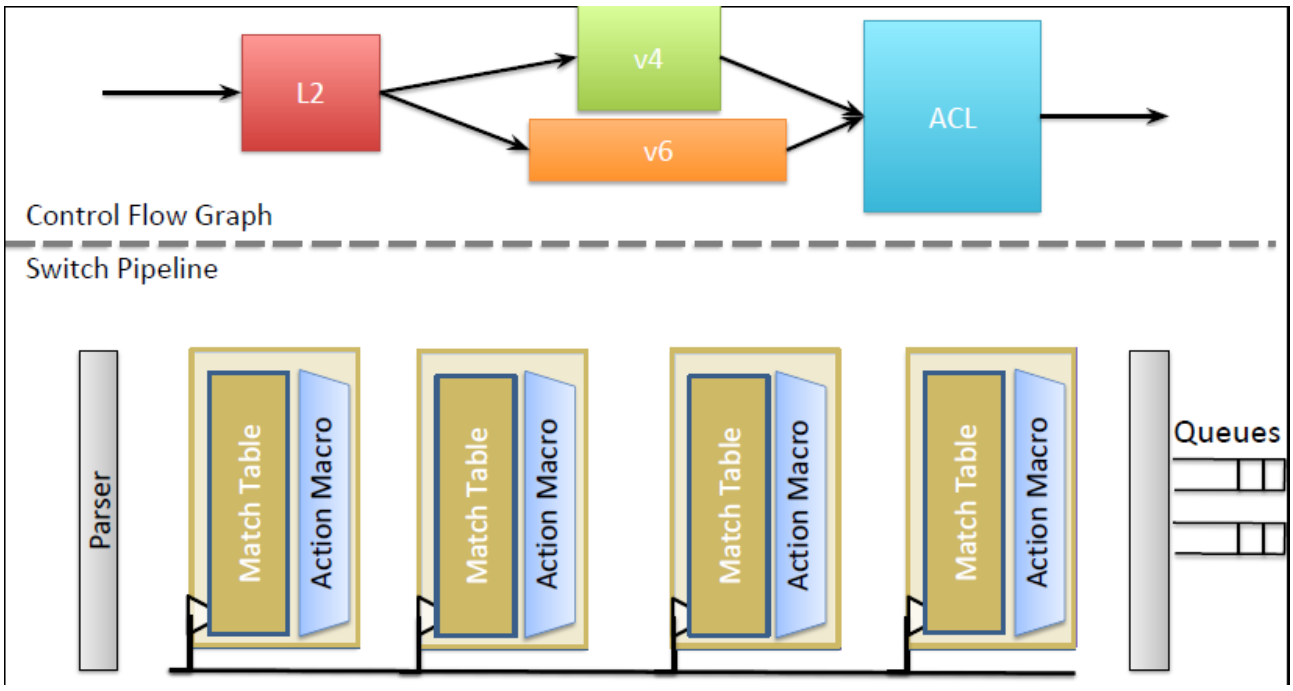


Figure 2.11: Programmable Switch Pipeline Chip [34]

The following are functions that can be performed on switches with PISA capabilities using P4 [34]:

- Each stage is identical (what it does + dimensions) and implements a match + action
- matching on the packet header (based on the protocol) in Memory
- action is performed on the ALU
- Initially it has no concept of what a protocol is, and the programmer writes this instruction
- The programmer must declare which headers are recognised
- The programmer declares what tables are needed and what actions to be processed
- All stages are identical at design

There are three main goals in designing P4 [10]:
1. Reconfigurability: The controller should be allowed to define packet parsing and processing field.
2. Protocol Independence: Switches should not depend on specific packet formats [10]. However, the controller should define packet parser for extracting header fields with specific types and names and gather packet types with the match-action table that processes these headers.
3. The programmer should not necessarily have to know the details of the target switch, but "a compiler should take the switch's capabilities into account when turning a target-independent description (written in P4) into a target-dependent program" [10].

The main goals of P4 language configuration are shown in Figure 2.11 below and followed by a subsection explaining each of the P4 switch configuration elements.

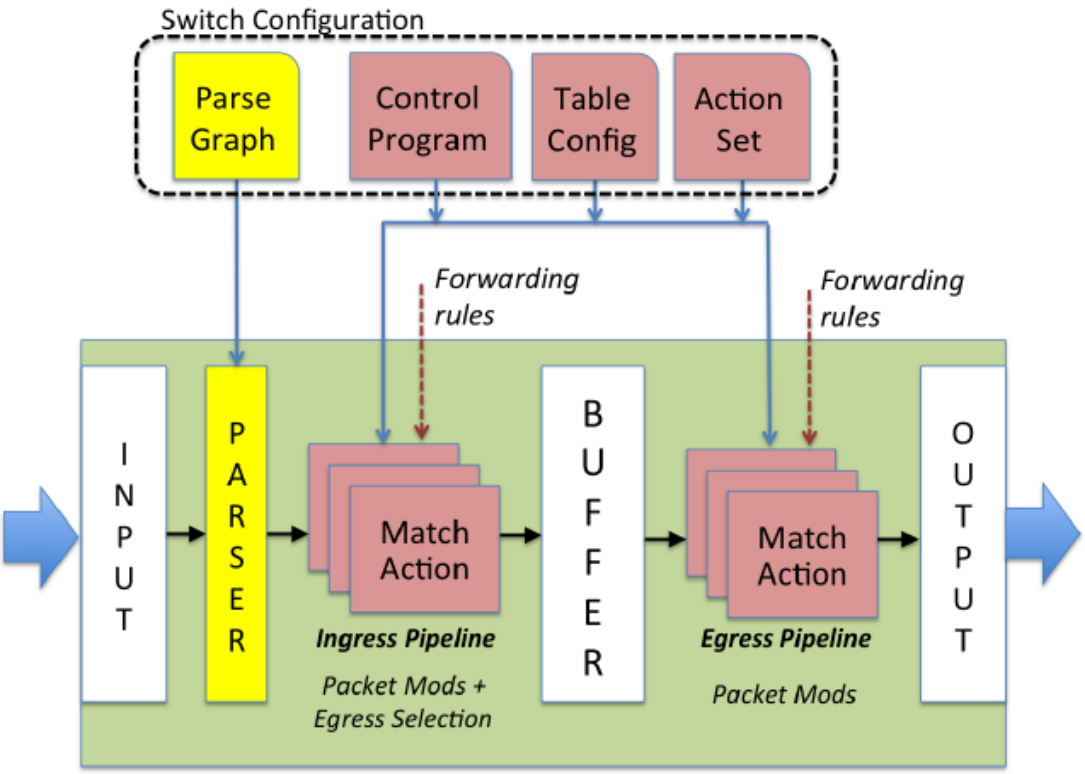


Figure 2.12: Switching Forwarding Model [10]

2.4.1 P4$_{16}$ Headers

Headers describes the structures and sequence of series of fields and specifies the width and constraints of field values [10]. Headers can be customized but the directed protocol must be specified. Below is sample code showing Header Types and Header Instances.

P4 Types can have simple header type definitions as follows [35]:

```
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;
header ethernet_t {
  macAddr_t dstAddr;
  macAddr_t srcAddr;
  bit<16>   etherType;
}
header ipv4_t {
  bit<4>    version;
  bit<4>    ihl;
  bit<8>    diffserv;
  bit<16>   totalLen;
  bit<16>   identification;
  bit<3>    flags;
  bit<13>   fragOffset;
  bit<8>    ttl;
  bit<8>    protocol;
  bit<16>   hdrChecksum;
  ip4Addr_t srcAddr;
  ip4Addr_t dstAddr;
}
```

Basic Types

- bit<n>: Unsigned integer (bitstring) of size n
- bit is the same as bit
- int<n>: Signed integer of size n (>=2)
- varbit<n>: Variable-length bitstring

Header Types: Ordered collection of members

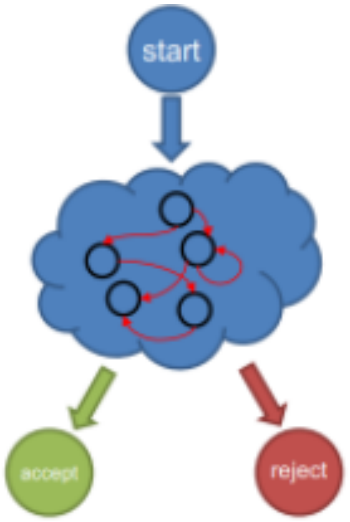
- Can contain bit<n>, int<n>, and varbit<n>
- Byte-aligned
- Can be valid or invalid
- Provides several operations to test and set validity bit: isValid(), setValid(), and setInvalid()

Typedef: Alternative name for a type

2.4.2 P4$_{16}$ Parsers

A Parser defines how to identify headers and valid header sequences within packets.

- Parsers are special functions written in a state machine style
- Parsers have three predefined states
    - start
    - accept
    - reject
        - Can be reached explicitly or implicitly
        - What happens in reject state is defined by an architecture



Other states are user-defined [34]

Below is Figure 2.12 showing an example of a P416 Parsers (V1Model). The right side of the figure shows that the parser in the V1Model accepts the input packet and must produce the parsed header representation. The user metadata and standard metadata buses also pass through the parser and can be used if desired [34].

```
/* From core.p4 */
extern packet_in {
  void extract<T>(out T hdr);
  void extract<T>(out T variableSizeHeader,
            in bit<32> variableFieldSizeInBits);
  T lookahead<T>();
  void advance(in bit<32> sizeInBits);
  bit<32> length();
}
/* User Program */
parser MyParser(packet_in packet,
          out headers hdr,
          inout metadata meta,
          inout standard_metadata_t std_meta) {

  state start {
     packet.extract(hdr.ethernet);
     transition accept;
  }

}
```

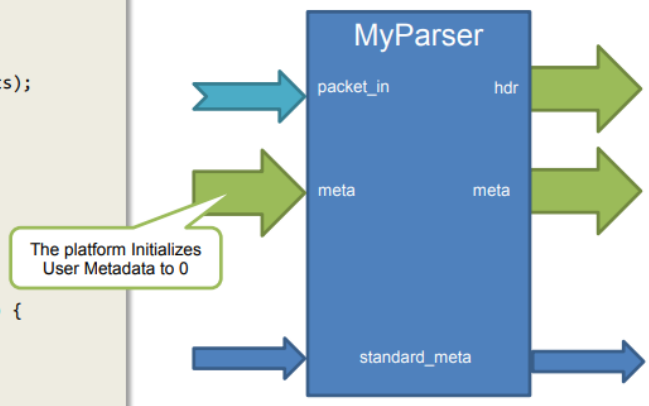


Figure 2.13: Parsers (V1Model) [35]

Top left shows the definition of the packet type from the core. p4. This type has various methods that are very useful when writing the parser. Lastly, the bottom is an example of a very simple parser that only defines the functionality of the start state. It simply extracts the ethernet header from the packet and then transitions to the 'accept' state. A complete state machine for implementing a parser is shown in Figure 2.13 below.

```
/* User Program */
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t std_meta) {

 state start {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.type) {
      0x800   : parse_ipv4;
      default : accept;
    }
  }

  state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition accept;
  }
}
```

```
/* From core.p4 */
extern packet_in {
  void extract<T>(out T hdr);
  void extract<T>(out T variableSizeHeader,
                  in bit<32> variableFieldSizeInBits);
  T lookahead<T>();
  void advance(in bit<32> sizeInBits);
  bit<32> length();
}
/* From v1model.p4 */
parser Parser(packet_in packet,
              out headers hdr,
              inout metadata meta,
              inout standard_metadata_t std_meta);

/* From simple_sume_switch.p4 */
parser Parser(packet_in packet,
              out headers hdr,
              out user_data_t user,
              out digest_data_t digest,
              inout sume_metadata_t smeta);
```

Figure 2.14: Parser State Machine Implementation [34]

2.4.3 P4$_{16}$ Controls

P4 controls are like C functions, however they are without loops and can declare variables, create tables, and instantiate externs [35]. The functions are code specified in the apply statement and they control the flow from one table to the next. The processing of packets can be expressible as DAG as follows [35]:

- Match-Action Pipelines
- Deparsers
- Additional forms of packet processing (updating checksums)

Below is a sample simple action function declared inside a P4 controls, however functions can be declared globally.

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {

  action swap_mac(inout bit<48> src,
                  inout bit<48> dst) {
    bit<48> tmp = src;
    src = dst;
    dst = tmp;
  }

  apply {
    swap_mac(hdr.ethernet.srcAddr,
             hdr.ethernet.dstAddr);
    std_meta.egress_spec = std_meta.ingress_port;
  }
}
```

- Parameters have type and direction
- Variables can be instantiated inside
- Supports arithmetic and logical operations (+,-,*,&,>>, ==, etc)

2.4.4 P4$_{16}$ Tables

Tables describes how the defined header fields are to be matched in the match-action stages, this can take a form of an exact, range or wildcard match and determines what actions need to be performed when a match occurs [10]. Figure 2.14 shows match-action processing performed by Tables in P4.

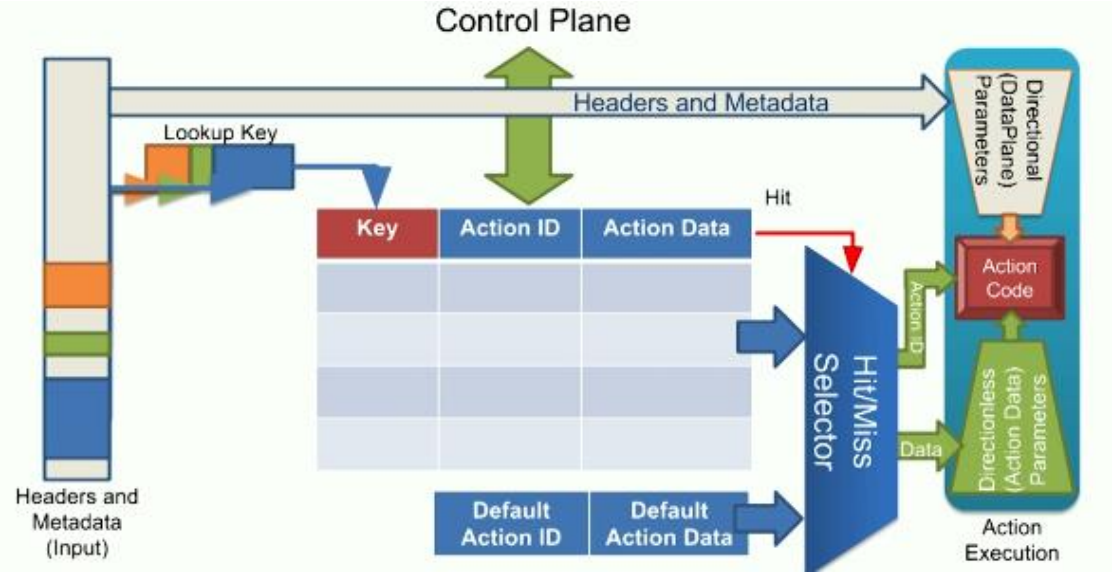


Figure 2.15: Match-Action Processing [35]

The following steps are performed in the match-action processing:

- Each table consists of a specified Key, Action ID and Action Data entries and can be added or removed from the Control Plane. Each table also specifies default action ID and action data.
- When a packet comes in, they extract the specified bits from the input headers and metadata to create the key and identify that key within the table entries. If a match is found within one of the entries, Action ID and Data process the corresponding entry. If a match is not found, default action ID and Data are used.
- Both the outputs are sent out to the multiplexer which chooses one of them based on whether a header was found on the table.
- The selected header ID and header Data send to the action execution unit.
- The action execution unit output and update headers and metadata are defined.

To an IPv4 router, a routing table must be defined as shown in the example below. The Figure 2.15 highlights the difference between Data Plane and Control Plane in tables.

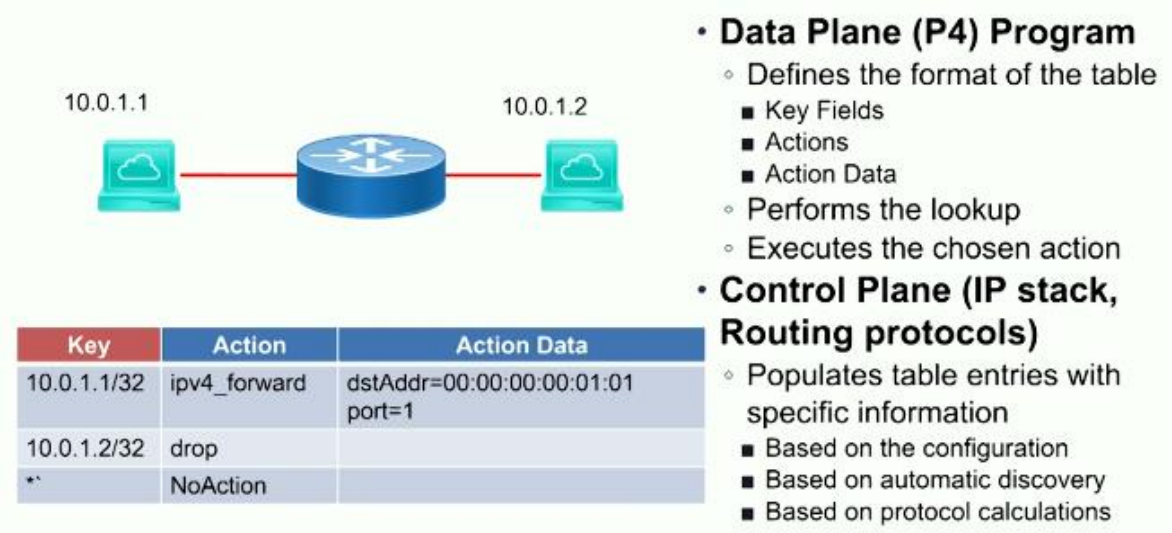


Figure 2.16: IPv4_LPM Table structure [35]

2.4.5 P4$_{16}$ Deparsing

The last stage of P4 switching is deparsing. Deparsers are responsible for assembling the headers back into a well-formed packet and makes deparsing explicit but couples from parsers. Below is Figure 2.16 showing the instructions performed for deparsing.

```
/* From core.p4 */
extern packet_out {
  void emit<T>(in T hdr);
}

/* User Program */
control DeparserImpl(packet_out packet,
                     in headers hdr) {
  apply {
    packet.emit(hdr.ethernet);
    packet.emit(hdr.ipv4);
    ...
  }
}
```

Figure 2.17: P4$_{16}$ Deparsing

2.4.6 P4Runtime

Figure 2.17 shows the architecture of the implementation of P4runtime. Once the P4 program written and compiled to a target-specific configuration binary, it is loaded to the data plane. Runtime allows control of all the entries of the P4 program and enables modification of defined tables of the data plane.

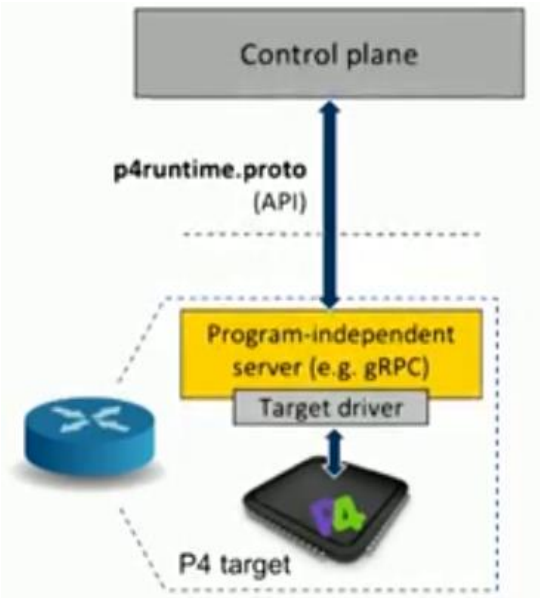


Figure 2.18: P4Runtime Model

P4Runtime is a framework for runtime control of P4 targets, which is an open-source API and server implementation. It is a protobuf based, which means one can easily generate code to serialize and deserialize

messages for different programming languages, uses gRPC for RPC framework which makes it easy to generate code for other programming languages using the control place. Since P4Runtime is program-independent, the API does not change when P4 program changes.

Below is Table 2.2 showing the advantage of P4 Runtime over existing approaches of runtime control.

API	Target-independent	Protocol-independent
P4 Compiler auto-generated	✔	✘
BMv2 CLI	✘	✔
OpenFlow	✔	✘
SAI	✔	✘
P4Runtime	✔	✔

Table 2.2: Properties of runtime control API

The table shows that every one of the other API approaches is either target-independent or protocol-independent and only P4Runtime is both target-independent and protocol-independent.

2.5 Related Work

2.5.1 OpenTSN

The recent standardization and rapid improvement of the TSN has presented the comprehensive standard system with wide variety of choices. Quan et al [13], highlights that there is a large gap between TSN standards and the application specific TSN systems which present even a better opportunity for researchers and designers. OpenTSN is an open source for time sensitive networking system development by Quan et al [13], which presents a standard easy-to-use standard prototype platform that can be used for wide and rapid evolving TSN systems. Their paper introduces an OpenTSN, which is aimed at being an open source project to be a wide scope of TSN system customization. Their project consists of three main features for TSN Systems:

- SDN-based TSN network control mechanism
- Time-sensitive management protocol
- Time-sensitive switching model

OpenTSN is designed to cater for all hardware and software source codes so that designers can quickly and flexibly use their applications. OpenTSN is set to support customization of TSN systems on FPGAs from different level, i.e. system, device, and module level [13]. The project is said to not only provide source codes, but aimed to an open source to documents, interfaces, workflow, and design parameters of modules. OpenTSN offers parameters that can be customized to support several features of IEEE802.1 standard.

An architecture of OpenTSN basic components (TSN-Switch, TSNNic and TSNLight) for constructing TSN networks is shown in Figure 2.19. TSN-Switch and TSNNic are responsible for switching data between ethernet ports and network adapters for modifying data between local CPU and the network [13]. The two components have transmission deterministic capabilities, with TSNSwitch supporting only FPGA logic and TSNNic with FPGA and CPU system support capabilities [13]. OpenTSN supports TSN unrelated functionalities such as, local management, packet parsing, switching and forwarding,

and TSN related standards such as, time synchronization, traffic shaping carried out by hardware TSNSwitch and TSNNic and system configuration performed by TSNLight. TSNLIght is a centralized network controller for OpenTSN to map streams on TSN networks [13].

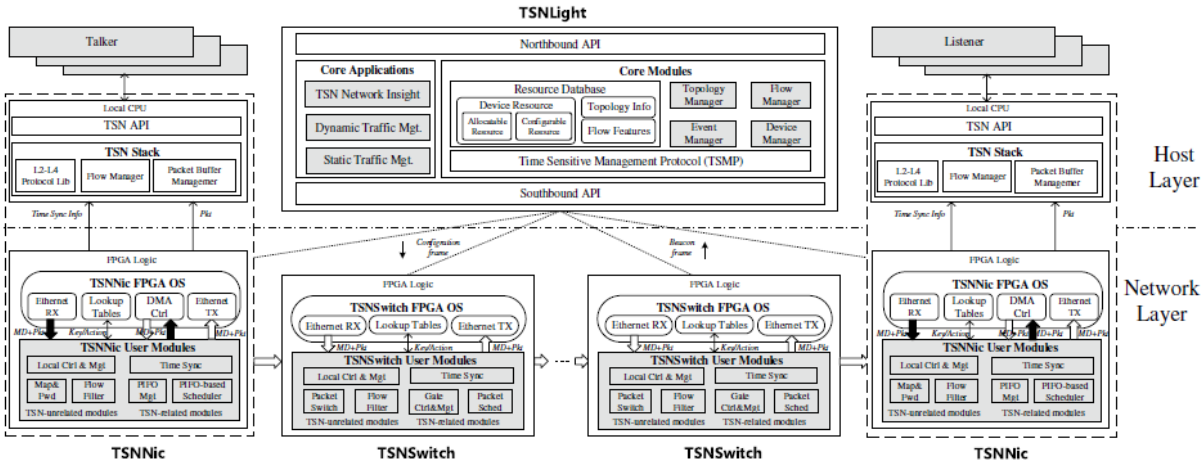


Figure 2.19: OpenTSN architecture [13]

The parts highted in grey in Figure 2.19 shows sections of the OpenTSN that can be changed by designers to suit their requirements. For example, to improve resource efficiency of the network resources, flow scheduling algorithms can be altered on the controller to suit new applications using northbound interface provided by TSNLight.
The following subsections will give a breakdown of the three features of OpenTSN.

SDN-based TSN network control mechanism

SDN based TSN network uses the proposed IEEE 802.1Qcc standard as the centralized control model for OpenTSN. The model is used for configuring a TSNSwitch and TSNNic network devices, as shown in figure 2.18. However, unlike the IEEE 802.1Qcc model in Figure 2.7, talkers and listeners in OpenTSN can be configured in central network configuration. With such configuration, TSNLight can effectively allocate network resources for streams that are unknown to the network in advance a table-miss event is triggered [13]. TSNLight capabilities allows the network controller to divide streams in three categories, namely, time-sensitive (TS) flows, rate-constrained (RC) flows and best-effort (BE) flows [13]. The three flows are what allows deterministic transmission and are basic features of TSN networking.

Time-sensitive management protocol

Time-sensitive management protocol (TSMP) act as channel of communication between the controller and TSN devices. Figure 2.20 below a compares TSMP OpenTSN protocol with IEEE 802.1Qcc protocol in their designated layer of the OSI network model, which is layer 2 [13]. This protocol consists of two message formats, Configuration (C) and Beacon (B) which are distinguishable in the PTP/PCF header section [13]. Both massages types control a flow of incapsulated packet information flowing from controller to devices and message (C) is also responsible to flow control of incapsulated information from device to controller. Each message class has multiple sub-class messages located in TSMP header [13]. Sub-class messages of (C) have registers such as switchtable_update, synfreq_update, sub-class (B) has message register such as state_report, table_miss, packet_sampling, etc. On the contrary, IEEE 802.1Qcc uses protocols such as NETCONF and RESTCONF to similar communications [13].

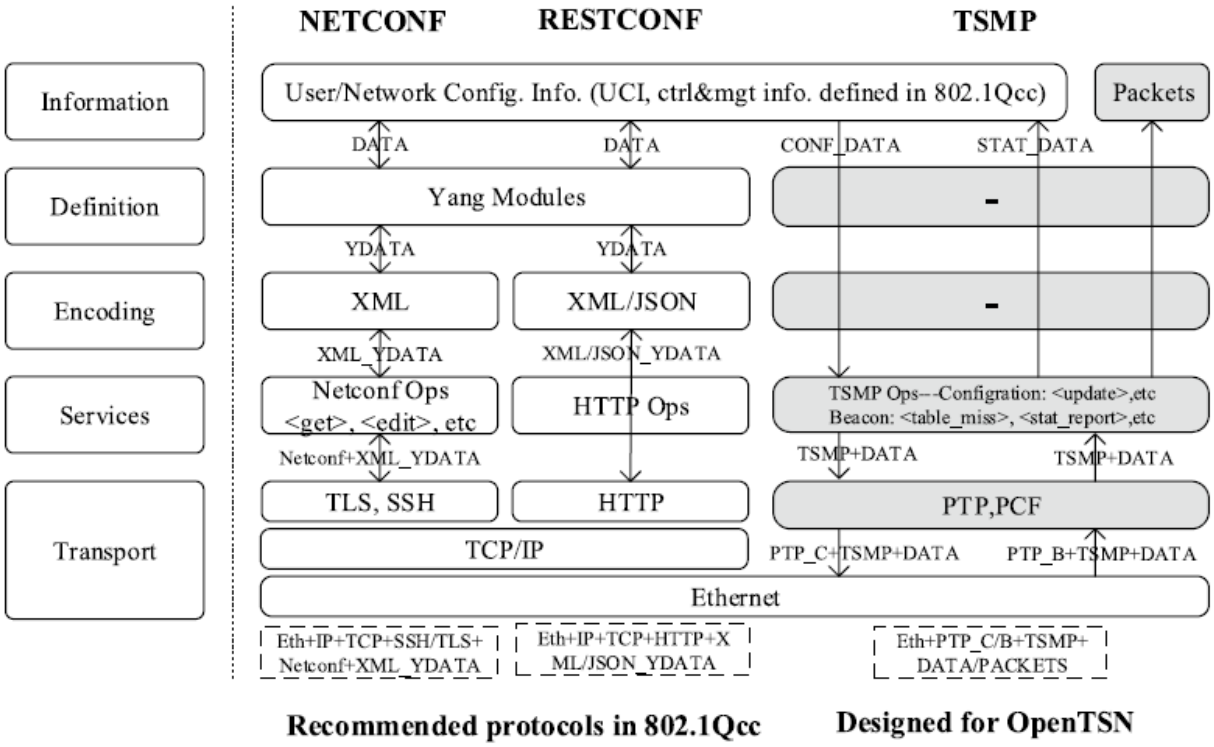


Figure 2.20: TSMP and protocols recommended in IEEE 802.1Qcc [13]

Time sensitive switching model

The time sensitive switching in OpenTSN is meant to perform deterministic transmission for TSN switches. A model of the switching is shown in Figure 2.21 and contains processing logics (A-G) and storages (Ingress queue, Describer queue, Packet buffer and Egress queue) [13].

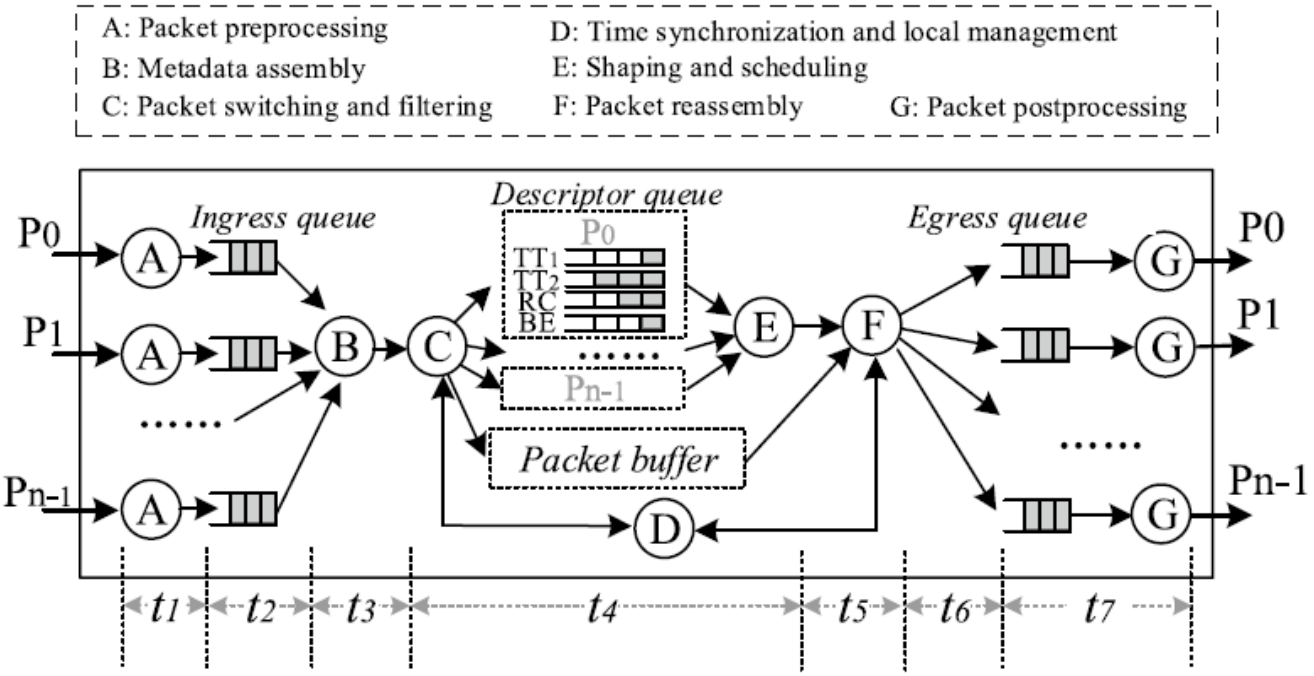


Figure 2.21: A time sensitive switching model [13]

Table 2.3 shows the logics in the switching model and the functions performed by each logic state.

Logic	Function
A	Responsible for packet preprocessing such as CRC checking and timestamping
B	Is an aggregator that schedules data from multiple queues using a round robin algorithm and a metadata before each packet
C	Forward synchronized and configured packets to logic D.
D	• Performs global network time synchronization processing and local modules control. • Forwarding of TS, RC, BE packets to packet buffers with their descriptors known by corresponding queues.
E	Controls dequeue gates and executes flow metering.
F	reads the corresponding packet of descriptor from packet buffers and forwards the scheduled packets to egress queues.
G	Performs CRC calculations, queue regulations and transparent clock processing

Table 2.3: Functions performed logics of time sensitive switching model

The functions performed by logics above results in traffic shaping and scheduling which achieves on of the key functionalities TSN standards which is deterministic forwarding of time-sensitive traffic.

This research project aims at designing a TSN switch using P4 and implemented on TSN network using SDN as network controller. OpenTSN resembles this study in so many ways because it consists of all the features. However, not all features of OpenTSN can be investigated in this project, the switch used here is mainly hardware, whereas this project is mainly software based. Lastly this research project aims at designing TSN switch using P4, which in this paper is closely resembled by time sensitive switching model.

3. Design Requirements

3.1 System Requirements

There are no TSN-Switch and Network user requirements explicitly stated for this research project, however TSN-Switch design using P4 and SDN-controlled TSN network requirements are identified to meet TSN Standards and the deliverables of this research project.

3.1.1 Basic Switch Routing

RM1-0-001	IP Router
Requirement	The switch should perform basic routing
Rationale	This allows the switch to support other classes such as nonreal-time class packets
Refined By	RM1-0-001
Verification	RM1-2-001

RM1-0-002	Tunneling
Requirement	Switch to have basic tunneling to support IP Routing
Rationale	This will define a new header type to incapsulate the IP packet
Refined By	RM1-0-002
Verification	RM1-2-002

RM1-0-003	Implement Control Plane
Requirement	Implement the control plane in TSN-Switch using P4
Rationale	The controller will count every packet to have knowledge of incoming and outgoing packets
Refined By	RM1-0-002
Verification	RM1-2-003

RM1-0-004	Switch Latency
Requirement	Use P4 to ensure that the switch has TSN switch latency capabilities
Rationale	This will set a bounded time within a switch to measure traffic in and of the switch
Refined By	RM1-0-002
Verification	RM1-2-004

3.1.2 P4Runtime-Controlled TSN Switches

RM2-0-001	Topology detection
Requirement	Available links should be detected using OpenFlow network
Rationale	The network controller should be able to detect bridges associates with network topology
Refined By	RM2-1-004
Verification	RM2-2-002

RM2-0-002	Host detection
Requirement	TSN controller to detect talkers and listeners in the network
Rationale	A controller of an OpenFlow network should be aware of new hosts with the ARP protocol. Each ARP packets received by an OpenFlow switch is copied and forwarded to the

	controller [31]
Refined By	RM2-1-004
Verification	RM2-2-002

RM2-0-003	Time-synchronization
Requirement	A common-time frame should be established between talkers, listeners, TSN switches and network controller
Rationale	A network topology should be able to perform IEEE 802.1AS time-synchronization TSN standard to be considered a TSN network, because it forms the basis on a TSN network
Refined By	RM2-1-004
Verification	RM2-2-002

RM2-0-004	Traffic Scheduling
Requirement	Should calculate and configure Gate Control List of all TSN-switches
Rationale	A TSN network should be able to provide credit-based shaping which determines which traffic class is given priority as per IEEE 802.1Qbv TSN standard
Refined By	RM2-1-001, RM2-1-003
Verification	RM2-2-001

RM2-0-005	Systems Configuration
Requirement	Controller to provide a set of implementations to manage and control the network globally
Rationale	This is to provide an increase Stream Reservation Protocol (SRP) to allow talkers and listeners to interact with the controller as per IEEE 802.1Qcc TSN Standard
Refined By	RM2-1-004
Verification	RM2-2-002

3.2 Functional Requirements

Given the system requirements in place, the following system functional requirements are identified to ensure that system requirements are met.

3.2.1 Basic Switch Routing

RM1-1-001	Sending and Receiving Packets
Requirement	Make use of packet generating (python scapy) and packet sniffing (wireshark) tolls to perform routing
Refines	RM1-0-001
Verification	RM1-2-001

RM1-1-002	myTunnel Header
Requirement	Modify the switch to perform routing using myTunnel header
Refines	RM1-0-002
Verification	RM1-2-002

RM1-1-003	P4Runtime
Requirement	Implements Control Plane and allow the controller to monitor various entities that P4 exposes and use them to make control decisions.

Refines	RM1-0-003
Verification	RM1-2-003

3.2.2 P4Runtime-Controlled TSN Switches

RM2-1-001	Strict Priority Queueing in Bmv2
Requirement	Priority queueing is an interface that allows implementation of IEEE 802.1Qbv time-aware shaper in a virtualized environment such as Mininet using Bmv2 switch to select a priority queue
Refines	RM2-0-004
Verification	RM2-2-001

RM2-1-002	Priority Metadata
Requirement	Intrinsic_metadata.priority must be enabled in the struct metadata standard in order to apply Bmv2 priority queueing in Mininet
Refines	No refines
Verification	RM2-2-001

RM2-1-003	Bmv2 Switch
Requirement	Since the TSN network will be implemented in Mininet using P4, the implementation of TSN-Switches is based on p4 behavioral model switch which supports time stamped packet TSN-traffic scheduling
Refines	RM2-0-004
Verification	RM2-2-001

RM2-1-004	P4Runtime
Requirement	TSN network will use P4runtime for implementation of a TSN-controller based on IEEE 802.1Qcc specifications.
Refines	RM2-0-005
Verification	RM2-2-002

3.3 Acceptance Test Protocol (ATP)

This section highlights the test and acceptance protocols that will be performed on both the TSN-switch design using P4 and the TSN network. The design must meet the specified requirements in order to pass the tests.

3.3.1 Basic Switch Routing

RM1-2-001	IP Routing
Tests	RM1-0-001, RM1-1-001
Description	• This test will be performed on a Mininet topology using the BMv2 software switches, one host needs to be a send.py and another receive.py. • The send.py switch to send a message to the receive.py host by specifying the destination address of the host.
Pass Requirements	The message in the receive.py switch must arrive correctly, by having an ethernet header, IP header and correct TTL.
Fail Requirements	Message arrives incorrectly

RM1-2-002	Tunneling
Tests	RM1-0-002, RM1-1-002
Description	• This test adds myTunnel tables with a default action being drop • Create different table entries for different possible values of myTunnel headers destination ID field • This test will be performed on a Mininet topology using the BMv2 software switches, one host needs to be a send.py and another receive.py • The send.py and receive.py this time will be modified so that they can send tunnel packets with myTunnel header encapsulating P4 header
Pass Requirements	• To pass the test, we send a tunnel packet specifying destination ID and needs to arrive at the destination with showing "myTunnel" header, Ethernet header and IP header • The destination host does not have to look at the IP address but destination ID
Fail Requirements	Packet arrives incorrectly (missing myTunnel header)

3.3.2 P4Runtime-Controlled TSN Network

RM2-2-001	Traffic Scheduling
Tests	RM2-0-001, RM2-0-002, RM2-0-004, RM2-1-001, RM2-1-002, RM2-1-003
Description	• To test traffic scheduling, two streams with different priority will be created • The two classes will be classified as high priority and best effort class • The talkers will generate frames for both class and the bandwidth allocated for each class will be calculated.
Pass Requirements	To pass this test, packets/frames with high priority class should be reserved more bandwidth than the best effort class.
Fail Requirements	Both classes getting the same bandwidth reservation.

RM2-2-002	P4Runtime
Tests	RM1-0-003, RM1-1-003
Description	This test runs myConroller.py and installs the ingress and egress rules
Pass Requirements	To pass test, this should read tunnel counters and there should be a ping every one second which sends traffic to switches and over the channel
Fail Requirements	If there is no counters or traffic sent every one second on the channel

RM2-2-003	Switch Latency
Tests	RM2-0-005, RM2-1-004, RM2-1-004
Description	• This test will measure the set-up time of the User/Network interface (UNI) • Just like in the switch Runtime test, listeners will request streams and talkers will send streams and the time between the request and response time are measured.
Pass Requirements	The expected rending time from request time should be in microseconds to suit TSN requirements
Fail Requirements	Latency is significantly larger.

4. Design and Implementation

This chapter will carry out the full model system design and implementation of P4 TSN-Switch and SDN-Controlled TSN network which aims at achieving the system requirements and research deliverables outlined in chapter 3.

4.1 P4 TSN-Switch Design

This section will carry out a TSN switch and network design. A brief outline of Programming Protocol-Independent Packet Processors (P4) switch is explained in chapter 2. Section 2.4. Figure 4.1 below shows a broad P4 switch system architecture design approach for this experiment, with a detailed system explained thereafter.

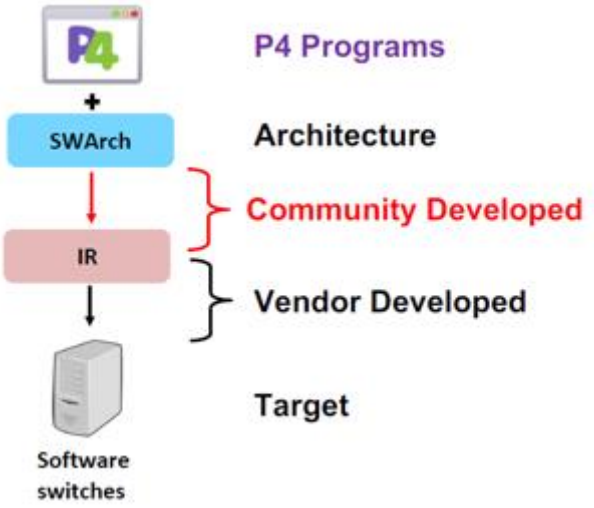


Figure 4.1: P4 TSN-Switch design Architecture

The breakdown of the design architecture will be carried in the top-down manner just like how todays network systems are built using the luxury of P4. P4 Program is as a language written to configure how packet should be processed in a switch. This is shown in the figure, it starts with a P4 program plus the architecture through the community and vendor developed to the target switch.

4.1.1 Model for Programming a P4 Switch

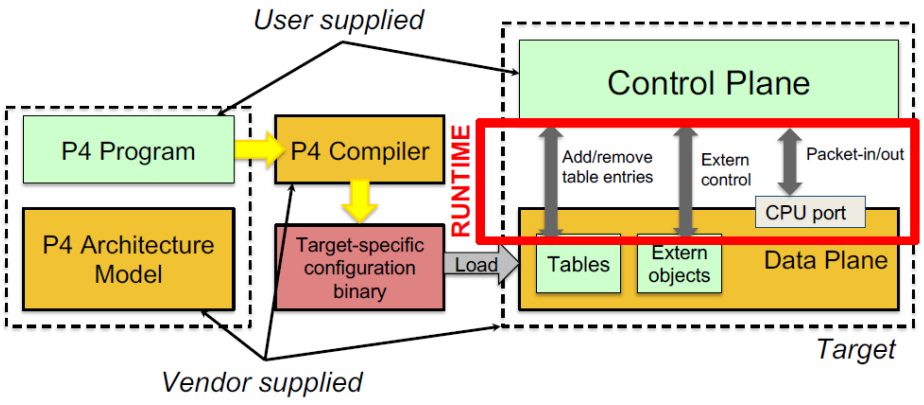


Figure 4.2: Model for Programming a P4 Target

As shown in Figure 4.2, a P4 programmer needs the elements provided by the vendor for packet processing to target and P4 architecture model to expose programmability of that target as well as P4 compiler. Then following tasks explain the programming of a target switch:

- Write the P4 program as user to implement the elements used in the architecture which may be parsers, control blocks, etc.
- Run my P4 program through the vendors compiler which will produce a target-specific configuration binary.
- This configuration will tell the target switch how to process the packets in the data plane.
- Provide programme for implementation of the control plane.
- The control plane and the data plane will interact through the RUNTIME mechanism which performs adding, removing, send and receive packets.

4.1.2 P4 Bmv2 Switch architecture

The P4 TSN-Switch model for this experiment is designed using Protocol-Independent Switch Architecture (PISA), which is a high-speed packet processing switch. PISA switch is convenient for TSN use because the high-speed packet processing capabilities. Below is Figure 4.3 showing a P4 BMv2 switch architecture PISA switch.

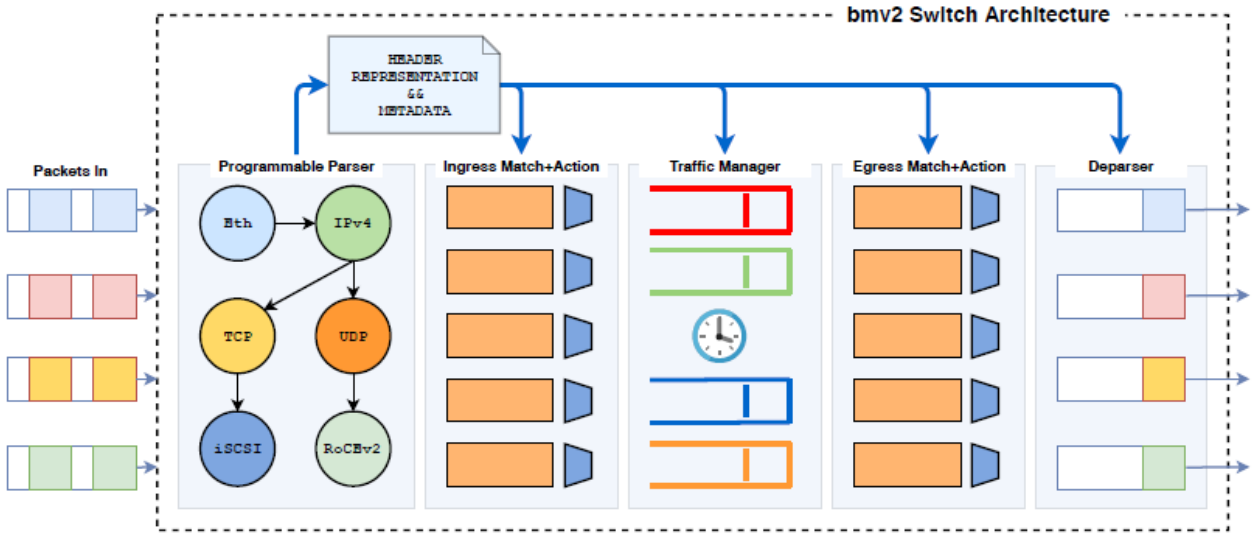


Figure 4.3: Architecture of P4 BMv2 Switch [38]

BMv2 is a protocol independent switch, and as seen in the figure, the switch model consists of the following stages: programmable parser, ingress match-action, traffic management, egress match-action and deparser. Parser is where to define the headers to be used and the order within the packet, this is then followed by the stages of ingress and egress match-action processing, which is where to define the tables and packet processing, and lastly the deparser which is puts back the headers back together to form the packet. The traffic manager between ingress and egress match-actions is responsible for packet queueing and scheduling of packet from ingress pipeline to egress pipeline.

4.1.3 Bmv2 Control Plane

TSN relies on synchronisation between hosts in the network to perform adequate scheduling, this requires an integration of SDN which allows synchronisation by introducing communication path management. TSN uses the concept of SDN to deploy control plane which populates the pipeline tables of stages of Figure 4.3 switch. Chapter 2, subsection 2.2.5 shows that the control plane can be applied

in a distributed (outside switch) or centralized (within switch) manner. Figure 4.4 shows BMv2 switch control plane configuration and its interaction with data plane entities using P4runtime.

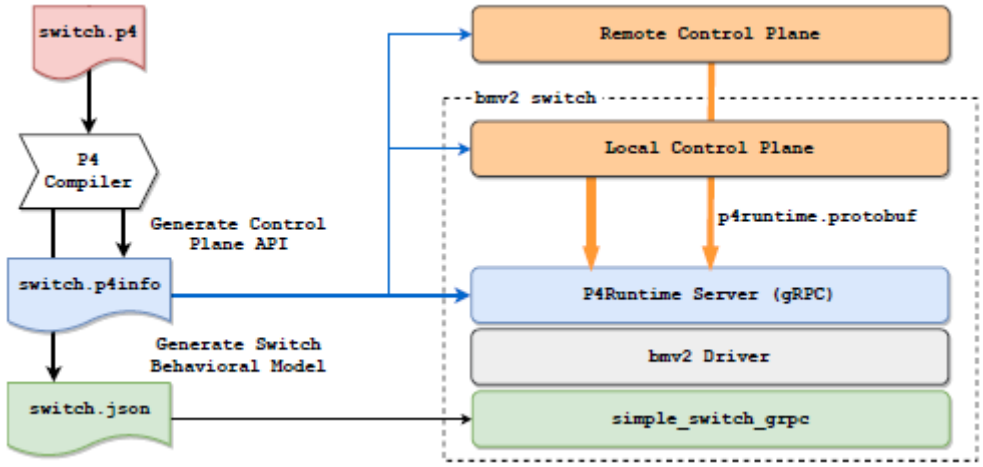


Figure 4.4: BMv2 switch Control Plane Configuration [38]

The figure shows that P4 BMv2 switch is equipped with both remote (outside switch) and local (within switch) control planes, this research project will however use the local control plane through p4runtime API to modify, add and make changes to the switch pipeline tables. Interaction of other of entities duplicate that of subsection 4.1.1.

4.1.4 TSN Network Design

Figure 4.5 shows a TSN network topology design under investigation, the topology is comprised of three TSN switches, five hosts and a local data plane P4Runtime controller. Each of the switches and the P4Runtime controller take form of what is outlined in section 4.1.

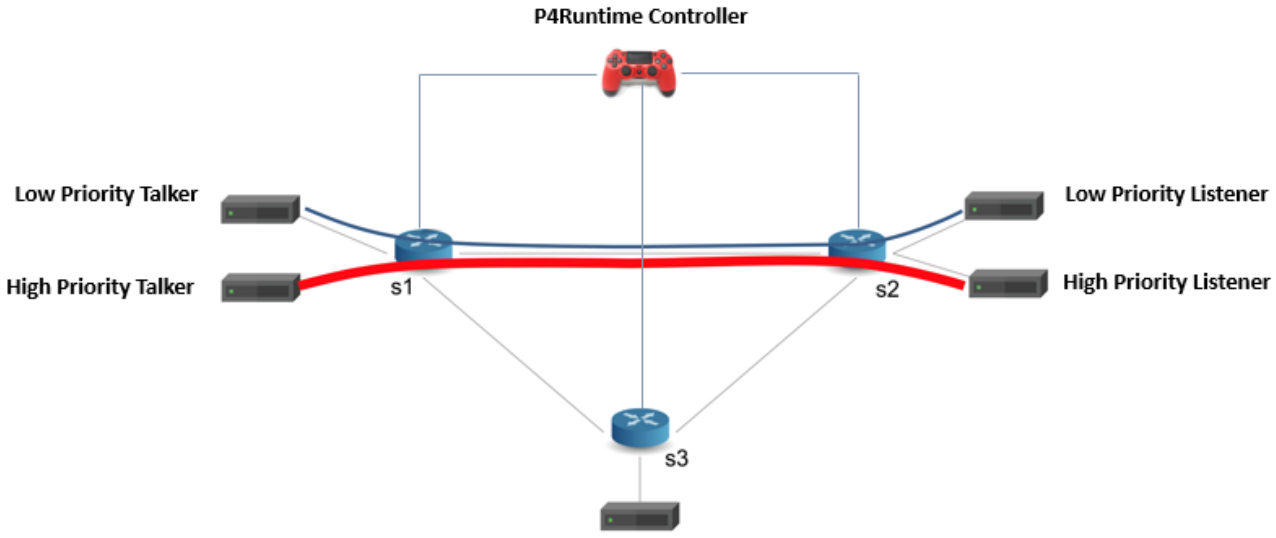


Figure 4.5: P4Runtime-Controlled TSN Network Topology

The network topology will be implemented on Mininet virtual emulator. The low priority talker will be named h1 and be assigned IP address 10.0.1.1 (mac addr: 00:00:0a:00:01:01), low priority listener will be named h2 and assigned IP address 10.0.2.2 (mac addr: 00:00:0a:00:01:02), high priority talker named h11 and assigned IP address 10.0.1.11 (mac addr: 00:00:0a:00:01:011), high priority listener named h22 and assigned IP address 10.0.2.22 (mac addr: 00:00:0a:00:01:022). The host attached to s3 will not be investigated but is named h3 and assigned IP address 10.0.3.3 (mac addr: 00:00:0a:00:01:03).

4.2 Implementation

This section provides implementation details of the TSN switch and SDN-controlled TSN network proposed in chapter 3. The implementation makes use of a code adapted from "The P4 language consortium, P4 tutorial" [40]. Each subsection includes header code listings of changes or additions made to achieve each of the requirements

4.2.1 RM1-0-001: Basic IP Router

This subsection implements all the models on the switch required to perform the basic IP routing.

- **Implementing Parser**

```
/*************************************************************************
********************** P A R S E R ***********************************
*************************************************************************/
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {
    state start {
        /* TODO: add parser logic */
        transition parse_ethernet;
    }
    state parse_ethernet {
          packet.extract(hdr.ethernet);
          transition select(hdr.ethernet.etherType) {
             TYPE_IPV4 : parse_ipv4;
             default : accept;
       }
    }
    state parse_ipv4 {
       packet.extract (hdr.ipv4);
       transition accept;
    }
}
```

Listing 4.1: IP Routing Parser

As shown in the parser program, the only added lines of code are highlighted under "TODO" field. The first step is to transition from the "state start" with command "transition parse_ethernet" because packets always start with the ethernet header. A different state is then defined for each header that has being invoked, and the states are "state parse_ethernet" and "state parse_ipv4". The TYPE_IPV4 is defined at the begging of the complete code which indicates the type of routing protocol supported. Inside the state parse ethernet state, incoming packet are invoked and the ethernet header is extracted. Furthermore, ethernet type field are examined by branching to IPV4 state or the accept state based on the value of the ethernet type field. If the type is ipv4, transition to parse ipv4 occurs, otherwise transition to default "accept" state to complete parsing.

- **Implementing Ingress Processing**

```
/*************************************************************************
************** I N G R E S S  P R O C E S S I N G  ***************
*************************************************************************/
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    action drop() {
        mark_to_drop(standard_metadata);
    }

    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port)
{
        /* TODO: fill out code in action body */
        standard_metadata.egress_spec =  port;
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    }
```

Listing 4.2: IP Routing Ingress Processing

This carries out an implementation of the ipv4 forward action. The action needs to do routing or set egress spec which is invoked by the action table which specifies the IP addresses and provide to this action the port numbers that the packet should go to.

The standard metadata field is where the traffic manager is told which port to send the packet to. Then move the packet current mac address to destination source address. Make the packet destination mac address to be the mac address that is provided when this action is invoked, this is also stored in the table entries. The last instruction documents the TTL field.

- **Implementing Deparser**

```
/*************************************************************************
*********************** D E P A R S E R  ***************************
*************************************************************************/
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        /* TODO: add deparser logic */
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}
```

Listing 4.3: IP Routing Derpaser

The deparser takes the packet “in” to be of type “packet_out” and then emit the headers by specifying the header type in the acceptable order. The order of emit is important and therefore, ethernet type is emitted before ipv4 type. The deparser will only emit the packet if is valid, which means if a packet does not have ipv4 header, the emit method will not be implemented.

4.2.2 RM1-0-002: Basic Tunneling

Tunning is known as a protocol that enables a secured forwarding of packet or data from one network to another. This is done by packet encapsulation which allows packet to seem like they are of public nature in the network, whereas the matter of fact they are private packet to ensure security. The following tasks will be performed to implement the switch basic tunning:

- Define myTunnel_t header type and to headers struct
- Update parser
- Define myTunnel_forwarding action
- Define myTunnel_exact table
- Update table application logic in MyIngress apply statement
- Update deparser
- Adding forwarding rules

- **Implementing the parser**

```
/*************************************************************************
********************** P A R S E R  ***********************************
*************************************************************************/

// TODO: Update the parser to parse the myTunnel header as well
        TYPE_MYTUNNEL:parse_myTunnel;
        TYPE_IPV4 : parse_ipv4;
        default : accept;
    }
  }

  state parse_myTunnel {
    packet.extract(hdr.myTunnel);
    transition select(hdr.myTunnel.proto_id) {
      TYPE_IPV4 : parse_ipv4;
      default : accept;
    }
  }
  state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition accept;
   }
}
```

Listing 4.4: Tunneling Parser

Here the switch to performs both basic IP routing and tunneling by adding another condition defined in Listing 4.4 as MYTUNNEL_TYPE and the existing IPV4_TYPE to have two types.
As shown on the second highlight, from IP routing, the parser logic has been updated with parse myTunnel state which extract the tunnel header. After the tunnel header is extracted, the myTunnel header contains a field called proto_id, the proto_id indicates what type of packet is encapsulated within the myTunnel header. If the proto_id is equals to the ipv4 defined above, then we want to transition to ipv4 state and this is done using the selection statement that enables transition between two states which based on transition proto_id. If proto_id is anything else, we terminate the parser with default accept statement.

- **Implementing Ingress Processing**

```
/*************************************************************************
************** I N G R E S S  P R O C E S S I N G  **************
*************************************************************************/
// TODO: declare a new action: myTunnel_forward(egressSpec_t port)
action myTunnel_forward(egressSpec_t port) {
    standard_metadata.egress_spec = port;
}

// TODO: declare a new table: myTunnel_exact
// TODO: also remember to add table entries!
table myTunnel_exact {
    key = {
        hdr.myTunnel.dst_id: exact;
    }
actions = {
            myTunnel_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = drop();
        }

    apply {
        // TODO: Update control flow
        if(hdr.ipv4.isValid() && !hdr.myTunnel.isValid()) {
            ipv4_lpm.apply();
        }

        if (hdr.myTunnel.isValid()) {
            myTunnel_exact.apply();
        }
    }
}
```

Listing 4.5: Tunneling Ingress Processing

The control logic ingress process is the same as that of the first implementation and can be seen in appendix A, and therefore it is not included in this listing.
Firstly, a new action called myTunnel_forward is defined, the purpose of this action will be to set the egress field appropriately so that the traffic manager knows which port to send the packet to.
Secondly, a new table called myTunnel_exact is defined. The table performs an exact match based on myTunnel destination ID field. Unlike in the first implementation were the key was ipv4 destination address, here the key for this table will be the myTunnel header destination ID field and we do an exact match. The new action invoked by this table is the myTunnel_forward action and the other actions are the same as the previous table.
Lastly, an update of the control logic. If the ipv4 header is valid and myTunnel header is not valid, then, and only then, we want to apply the ipv4 lpm table and perform IP routing. If myTunnel header is valid, then we want to forward packets based on myTunnel header and apply myTunnel exact table.

- **Implementing Deparser**

```
/*************************************************************************
*********************** D E P A R S E R  ***************************
*************************************************************************/
        // TODO: emit myTunnel header as well
        packet.emit(hdr.myTunnel);
```

Listing 4.6: Tunneling Deparser

4.2.3 RM1-0-003: Control Plane Implementation using P4Runtime

The final step of implementing P4 TSN-Switch is to showcase and implement the runtime control of the P4 data planes, detailed concept of P4Runtime are explained in chapter 2, subsection 2.4.6 and can also be seen in the Figure 4.2. The topology for this implementation is shown in Figure 4.6, with an added mycontroller.py.

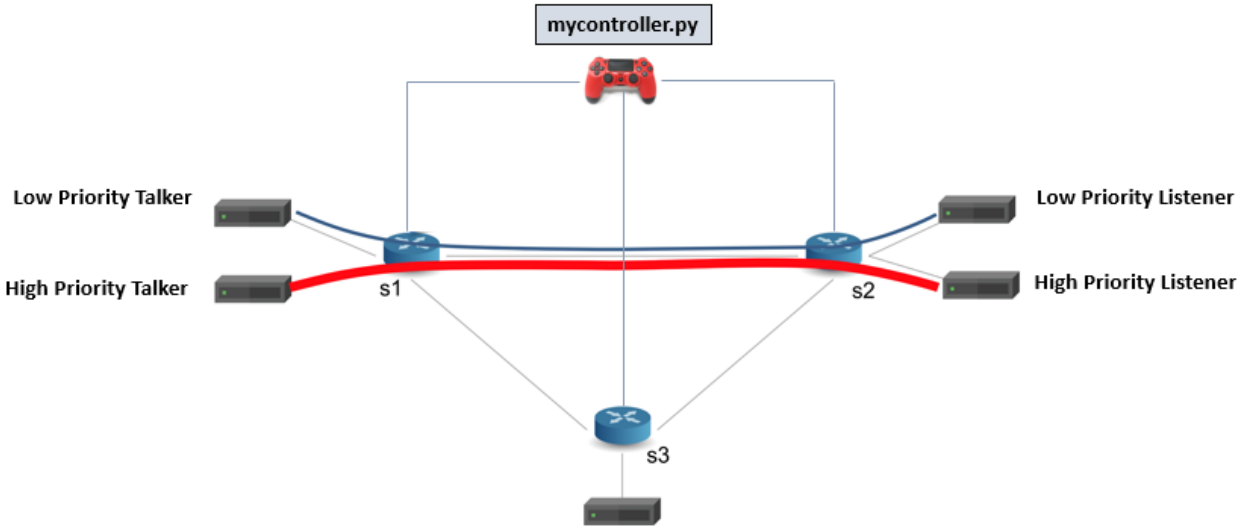


Figure 4.6: P4Runtime Mininet Topology

Unlike the previous implementations, this implementation will take the control piece out of the BMv2 CLI to the python script which will act as remote controller. The responsibility of the controller in this case will be to establish a gRPC connection to the switch for the P4Runtime service and to push the P4 programme to each switch and performs functions required to meet IEEE 802.1Qcc specifications.

- **Implementation of Tunnel Ingress Rule**

```
# TODO: build the transit rule
# TODO: install the transit rule on the ingress switch

  table_entry = p4info_helper.buildTableEntry(
      table_name="MyIngress.myTunnel_exact",
      match_fields={
          "hdr.myTunnel.dst_id": tunnel_id
      },
      action_name="MyIngress.myTunnel_egress",
      action_params={
          "dstAddr": dst_eth_addr,
          "port": SWITCH_TO_HOST_PORT
      })
   egress_sw.WriteTableEntry(table_entry)
   print "Installed egress tunnel rule on %s" % egress_sw.nam
```

Listing 4.7: Controller Tunnel Ingress Rule

As indicated above, the implementation here takes place in python script rather than using P4. Everything else like establishing a connection from mycontroller to the switches, setting of forwarding devices and setting of 2 tunnels has already been instantiated and are invoked in the main function. The entire python script can be found on github repository.

The code to be implemented is the transit rule. The first thing would be to target the correct table, and the table is myTunnel_exact. In this table, the header field to match is the myTunnel.dst_id, and since is an exact match, tunnel_id is provided. The action to be performed is myTunnel_forward, which takes a form of a "port" and this is written to the ingress of the switch.

4.2.4 TSN Network Implementation using Strict Priority Traffic Shapping

This implementation provides solutions to system requirements RM2-0-001, RM2-0-002, RM2-0-003 and RM2-0-004 in chapter 3, which ensures that the network achieve the TSN operational specifications.

BMv2 switch extensions

The BMv2 original files does not includes all the metadata required to carry some of the experiments, the following changes must be made in the v1model.p4 file:

- To implement priority queueing in the switches, we need to enable "#define SSWITCH_PRIORITY_QUEUEING_ON" which is a command in the v1model.p4 of the simple_switch.h that is compiled to enable priority queueing.

- In the v1model.p4 file, we must add the following metadata command "@alias("intrinsic_metadata.priority") bit<3> priority;" this command reserves the last three bits of the type of service to determine or set priority. Figure 4.7 below shows the IPv4 header field code, ToS and DSCP to highlight the changes.

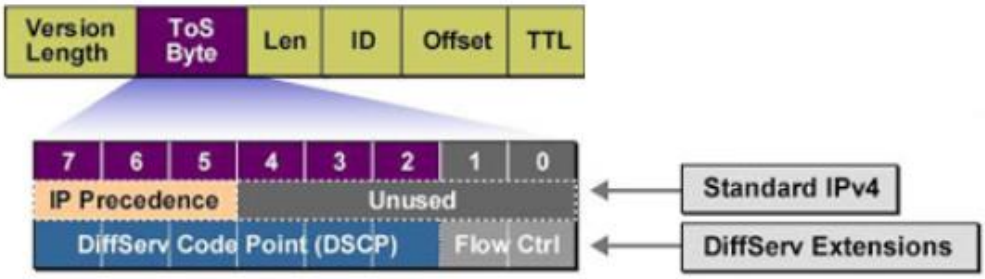


Figure 4.7: IPv4 header and ToS

- As shown in the figure, the whole 8 bits of the type of service will be allocated to the Differential Service Code Point.
- The three most significant bits of Differential Service Code Point will be allocated to priority decision (bit<3> priority;).

Implementation Rules

- **Defining Type of Service**

```
/*************************************************************************
*********************** H E A D E R S ***********************************
*************************************************************************/
 header ipv4_t {
    bit<8>    diffserv;
}
```

Listing 4.8: Differential Service Code Point

Having added the extensions of IPv4 type of service to the BMv2 switch as indicated above, differential service code point is defined in the header of the P4 code.

- **Implementing Ingress Processing**

```
/*************************************************************************
************** I N G R E S S  P R O C E S S I N G  ***************
*************************************************************************/

 apply {
      if (hdr.ipv4.isValid()){
          ipv4_lpm.apply();
          if (hdr.ipv4.srcAddr == 0x0a000101) {
              standard_metadata.priority = (bit<3>)0;
          }

          else if (hdr.ipv4.srcAddr == 0x0a00010b) {
              standard_metadata.priority = (bit<3>)7;
          }
      }
  }
```

Listing 4.9: Priority Queueing Ingress Processing

In the BMv2 switch, the above ingress processing rules are applied to say, if the incoming packets have ipv4 header, the following forwarding rules are applied:

- If the incoming packets are from a source with a mac destination address with last three bits of the address being 101 (h1), assign it priority 0.
- If the incoming packets are from a source with a mac destination address with last three bits of the address being 10b (h11), are given priority over other packets and are assigned priority 7.

The logic of priority 0 and priority 7 can be related to chapter 2, subsection 2.2.2 IEEE 802.1Qbv Traffic Scheduling, which shows that a TSN switch would have 8 (0,1,2….,7) queue gates which can be used to determine traffic priority for queueing. In the case of BMv2 switch, priority is arranged such gate 0 would have the least priority and gate 7 would be given the most priority.

5. Results and Discussion

This chapter describes results setup environment and outlines the tests performed for systems designed and implemented in chapter 4. The results are then evaluated and a discussed on whether the system performance meet the system requirements in chapter 3 and that of the research project deliverables.

5.1 Results Setup

This section presents the tools used in setting up the experiment to obtain the results.

5.1.1 Mininet

The results of the experiment are obtained using Mininet emulator. Mininet is a network emulator used for purposes such as research and learning to complete experiments of networking on a laptop or PC. Mininet creates virtual switches, controllers, and links which run on a Linux network system, supports OpenFlow and SDN [39].

Mininet is Linux kernel dependent system and can create OpenFlow switches, controllers to control switch to allow communication between the hosts in a simulated network environment. “Hosts are represented as multiple emulated physical machines sharing the same functionality as the machine on which Mininet is deployed and can be accessed via Mininet CLI or API to perform shell commands.” Mininet is equipped with xterm, which is a standard terminal emulator for X windows System. The xterm allows the user to run experimental independent input/output processes and display the results.

5.1.2 Iperf

Measurements of TSN network performance is carried out using iperf. Iperf is an active measurement tool of IP networks and allows client and server to establish network protocols such as TCP, UDP, SCTP with IPv4 and IPv6 [40]. Iperf allows the following TCP/UDP measurement features:

TCP

- Bandwidth
- MSS/MTU size
- TCP window size

UDP

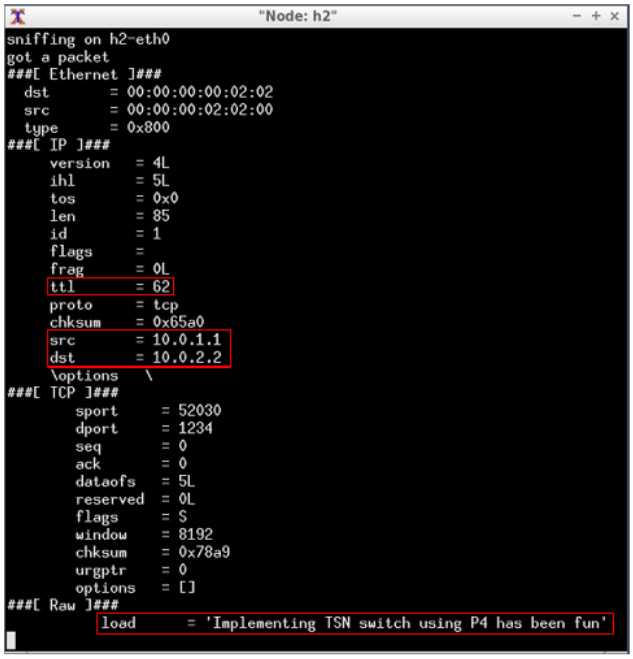
- Client can create UDP streams of specified bandwidth (througthput).
- Packet loss
- Delay jitter
- Round-trip time (RTT)
- Latency

5.2 Results and Discussion

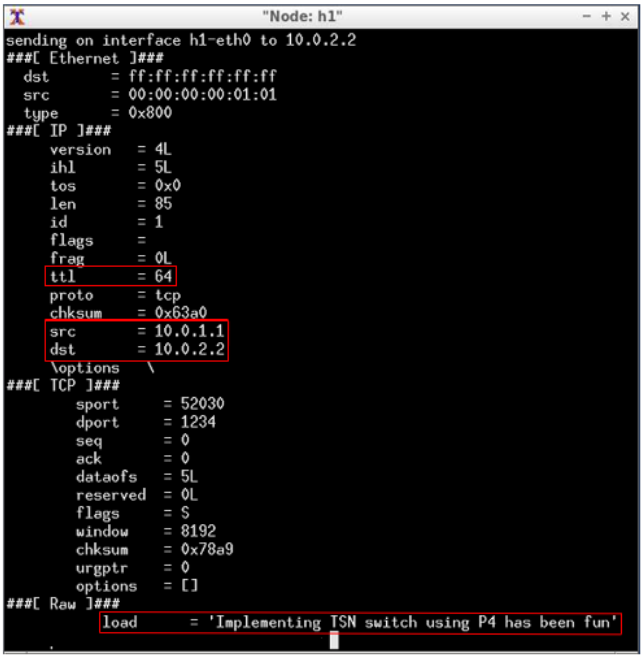
This section presents the results and discussion of tests performed to achieve to achieve the requirements of acceptance and tests protocol. Tests are performed using Mininet.

5.2.1 Basic IP Routing

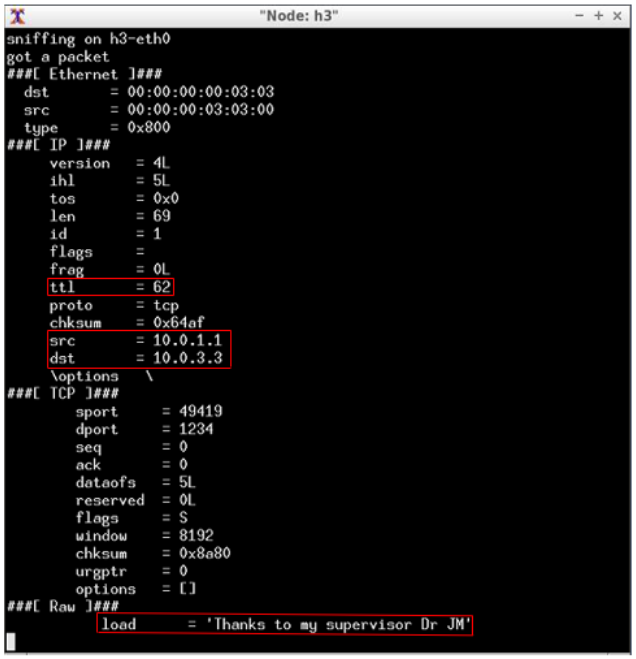
The results of this test are shown in Figure 5.1.



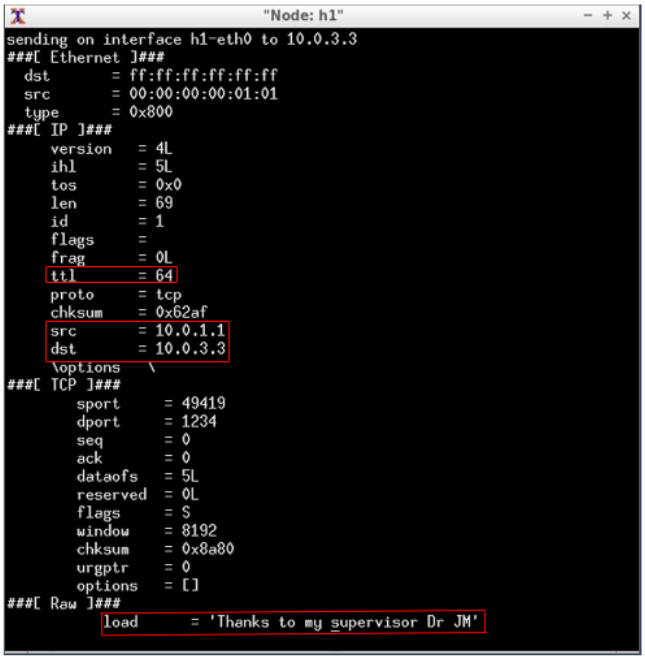
(a) h2 packet sniffing and message arrival



(b) h1 initializing connection and sending message to h2



(c) h3 packet sniffing and message arrival



(d) h1 initializing connection and sending message to h3

Figure 5.1: Basic IP Routing Results

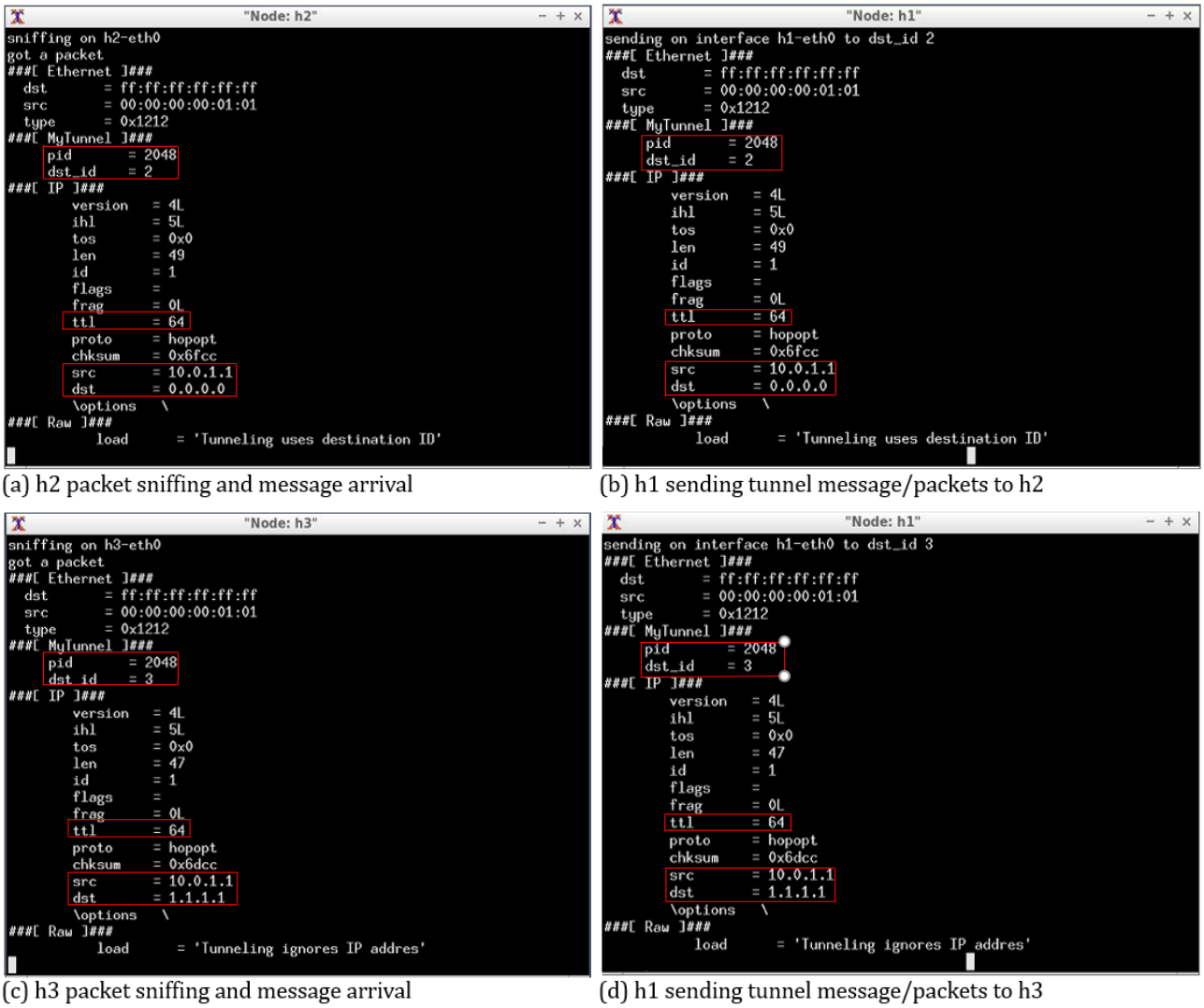
The results in Figure 5.1 show that before host 1 sends the message either, both host 2 and host 3 are only sniffing for packets, after host 1 sends the message, each of the destination hosts will acknowledge a received message with “got a packet”.

In (a) we see that host 1 sends a message to host 2 through a specified IP destination IP address: 10.0.2.2 and with message “Implementing TSN switch using P4 has been fun”, and in (c) we can see that the same message has arrived correctly.

Similarly, in (b) host 1 sends a message to host 3 with IP address: 10.0.3.3 with the message “It’s even better when your supervisor is Dr JM” and the message arrived correctly.
The message “Implementing TSN switch using P4 has been fun, it’s even better when your supervisor is Dr JM” has arrived correctly with correct IP headers and ttl, and therefore the acceptance test protocol RM1-2-001 requirement have been met.

5.2.2 Tunneling

The results of this test are shown in Figure 5.2.



(a) h2 packet sniffing and message arrival

(b) h1 sending tunnel message/packets to h2

(c) h3 packet sniffing and message arrival

(d) h1 sending tunnel message/packets to h3

Figure 5.2: Tunneling Results

Figure 5.2 show that the results of this experiment have a new header called the myTunnel which indicates that packets are forwarded using customized tunneling and not IP routing. (b) and (d) shows that, unlike the first test results, the sending host does not use destination address (10.0.2.2/10.0.3.3) to send packets, but uses the destination ID. The results show that packets are send to destination ID 2 and 3, which are destination ID of host 2 and host 3.

Furthermore, the ttl of the hosts h2 and h3 did not change or decrement, this due to the ability of tunneling to encapsulate private network packets travelling through public network to leave no trace for data security.

The messages have arrived correctly in each case, proving that acceptance test protocol RM1-2-002 requirements have been met.

5.2.3 P4Runtime

The results of this test are shown in Figure 5.3.

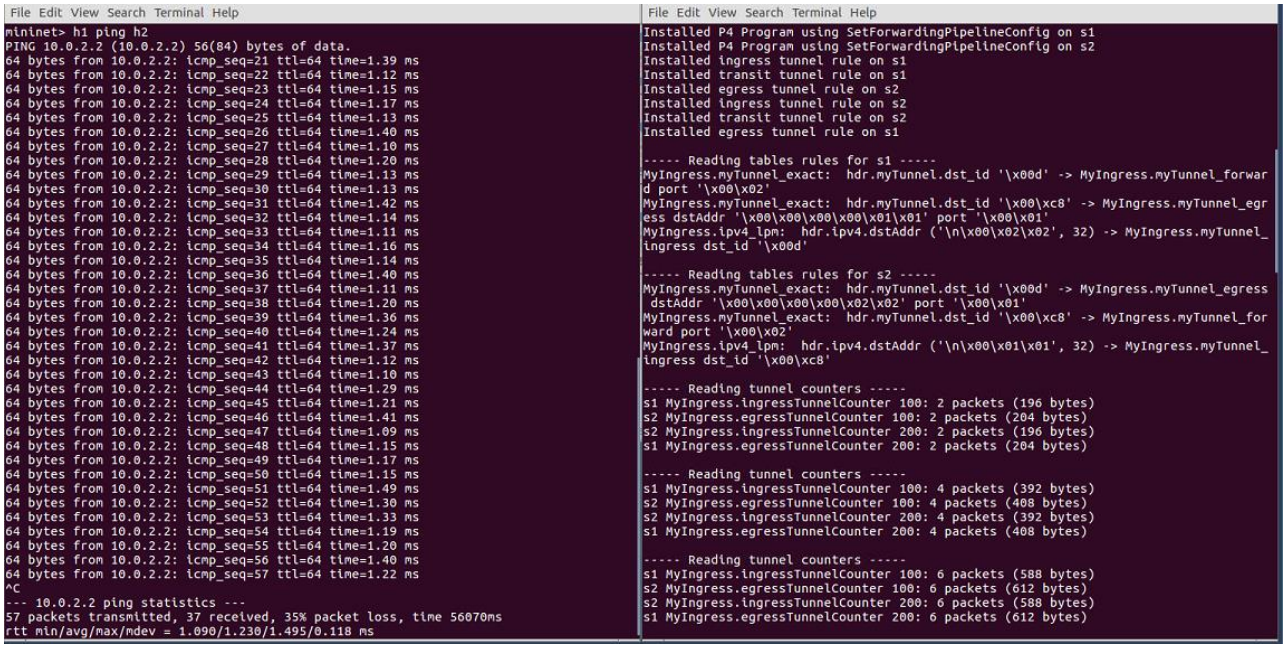


Figure 5.3: P4Runtime Results

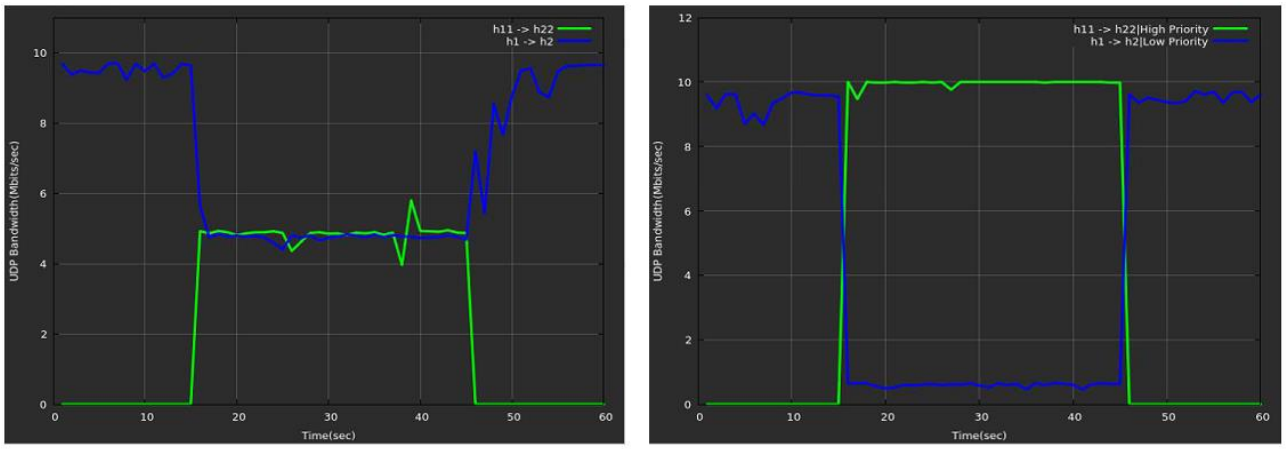
The results are obtained by performing a ping of h1 and h2, the packets are not immediately sent, it is after the tunnel rules were install that packets were send. The left window shows that there is a ping every one second, to send traffic over the tunnels.
The right window shows that ingress tunnel rules are installed, and the controller reads the tunnel counters. Since the ping sends traffic every one second, every two seconds the counters are counting the packets in the tunnels and the counters are incrementing. This way the controller manages every traffic in the network and uses this information to control the network.

The implementation works properly, and acceptance test protocol RM1-2-003 requirement are met.

5.2.4 P4Runtime-Controlled TSN Network

This test compares the results of a TSN network to an ordinary network and the advantages of using Time Sensitive Network switches to allow real-time applications to have priority over other applications. The test investigates bandwidth, packet loss, jitter and switch latency.

Bandwidth



(a) UDP Bandwidth with Priority Disabled (b) UDP Bandwidth with Priority enabled

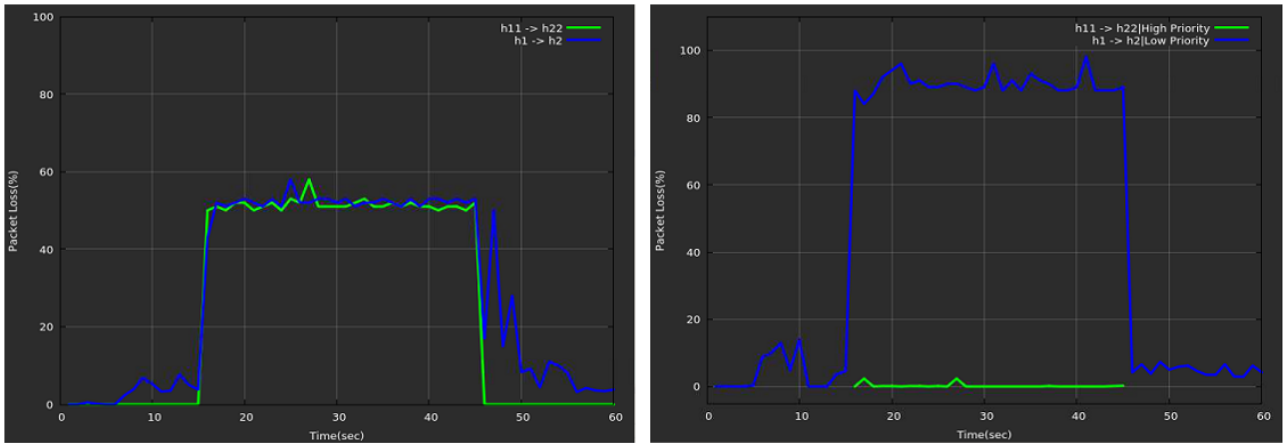
Figure 5.4: UDP Bandwidth Allocation

Bandwidth is a measurement of how much data can be transmitted over an internet network connection at a particular time. This implies that the connection can only handle the amount of data available on the link, the network link has maximum bandwidth of 10 Mbits/sec.

Figure 5.4. (a) shows that when the network operates normally and priority is not enabled, communication between h1 and h2 is allocated the maximum bandwidth of approximately 10Mbits/sec for the first 15 seconds when there is no connection established between h11 and h22. After 15 seconds when connection between h11 and h22 is established, the communication bandwidth between h1 and h2 is dropping to about 5Mbits/sec and the bandwidth of h11 and h22 goes to about the same amount of 5mbits/sec. This is because fair queueing is established whenever there are limited bandwidth resources in a link, and therefore both communicating links share the available bandwidth.

On the other hand, Figure 5.5 (b) shows the results when the network is operating as TSN network and priority is enabled to allow communication between h11 and h22 to priority over h1 and h2. The first 15 seconds a similar scenario to that of (b), where h1 and h2 are connected and allocated maximum bandwidth, however, when priority is enabled, when h11 and h22 establish a connection, this connection is given priority and allocated all the bandwidth resources and starving h1 and h2. The last 15 seconds shows that h1 and h2 connection is allocated bandwidth after connection between h11 and h22 is complete.

Packet Loss

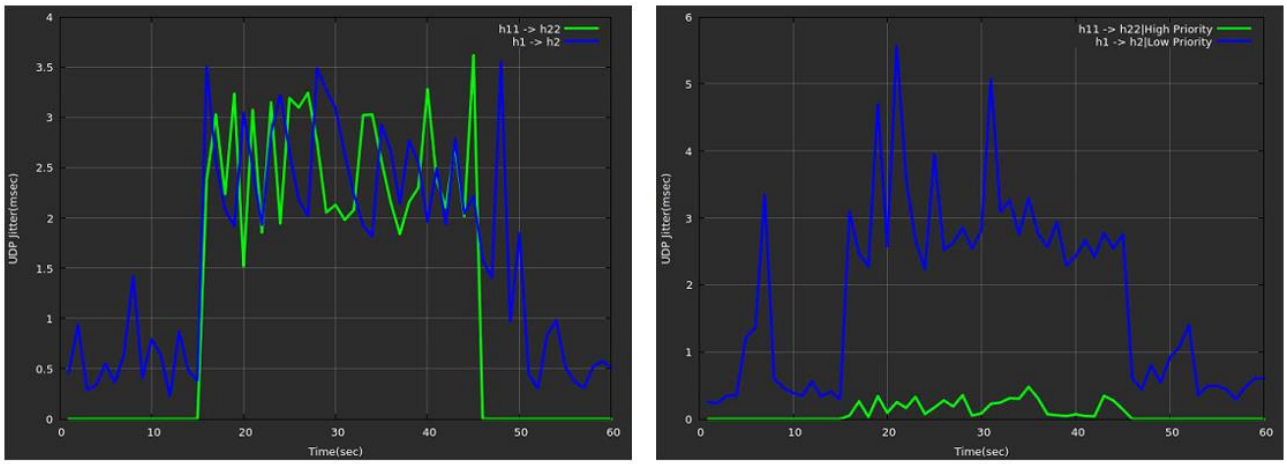


(a) UDP Packet Loss with Priority Disabled (b) UDP Packet Loss with Priority enabled

Figure 5.5: UDP Packet Loss

The results are obtained using the same tests done in the bandwidth allocation. Figure. 5.5 (a) shows that in the first 15 seconds, packet loss is below 10%, but as soon as communication between h11 and h22 is established, packet loss for around 50% which is due to available bandwidth.
When priority is enabled in the network, about the same network packet loss is observed in the first 15 seconds between h1 and h2 connection of around 10%. When priority hosts establish a connection, we observe a spike in packet loss of h1 and h2 communication and is around 90%. However, packet loss between high priority hosts is around 0% with few losses matching bandwidth allocation in Figure 34. After a complete data transfer of high priority hosts, low priority hosts packet loss does back to minimal loss.

Jitter



(a) UDP Jitter with Priority Disabled (b) UDP Jitter with Priority enabled

Figure 5.6: UDP Jitter

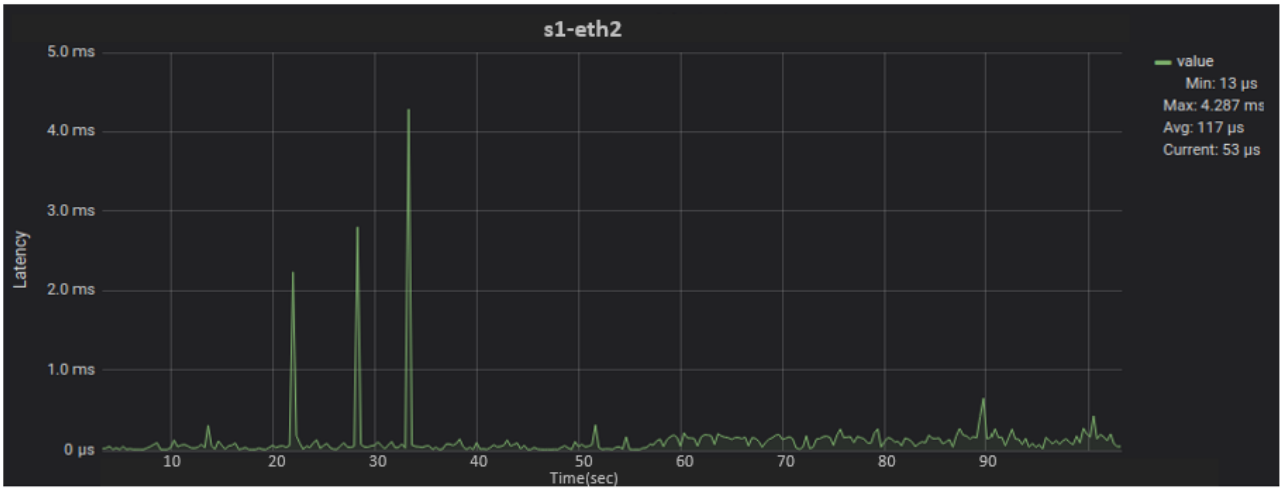
Similarly, Figure 5.6 presents the results of UDP Jitter obtained in two network modes of operation. Jitter is measurement of packets received in the order they were not sent at. Jitter is not desirable in real-time application like audio, because a substantial jitter would cause choppy audio.

The network mode where priority is not enabled, one can observe a similar trend as shown in the first two results. There is less jitter in the first 15 seconds when there is only communication between h1 and h2, there is more and about the same amount when there is communication in both links. When

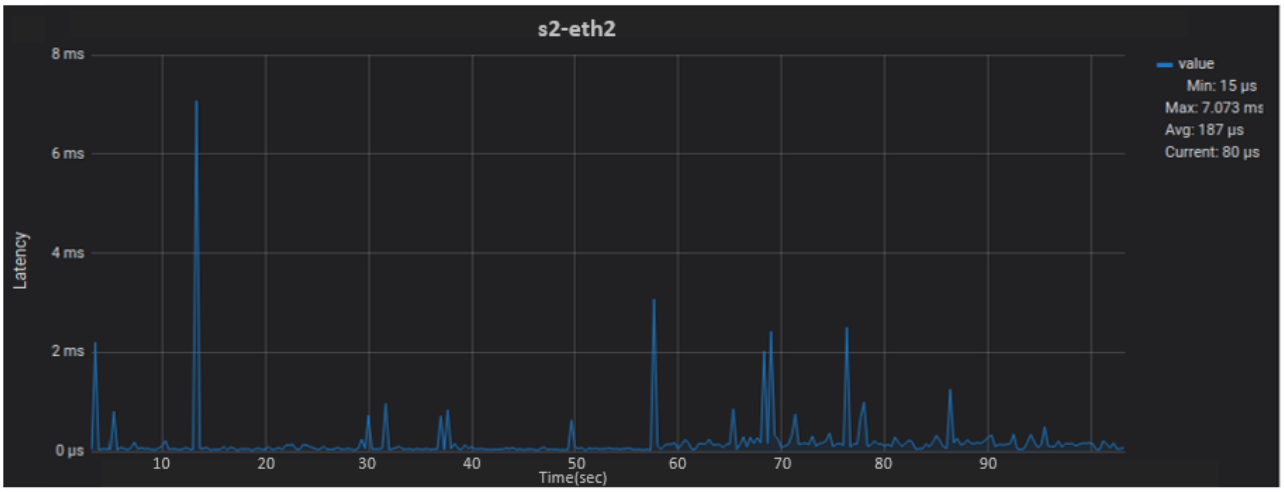
priority is enabled, the high priority channel experiences a minimal jitter and more jitter on the low priority communication channel.

This test used bandwidth, packet delay and jitter to show that communication between high priority talker h11 and high priority listener h22 is given priority over hosts h1 and h2 which are low priority talker and listener, and proves that acceptance and test protocol RM2-2-001 requirement is met.

Switch Latency



(a) Latency on Switch 1

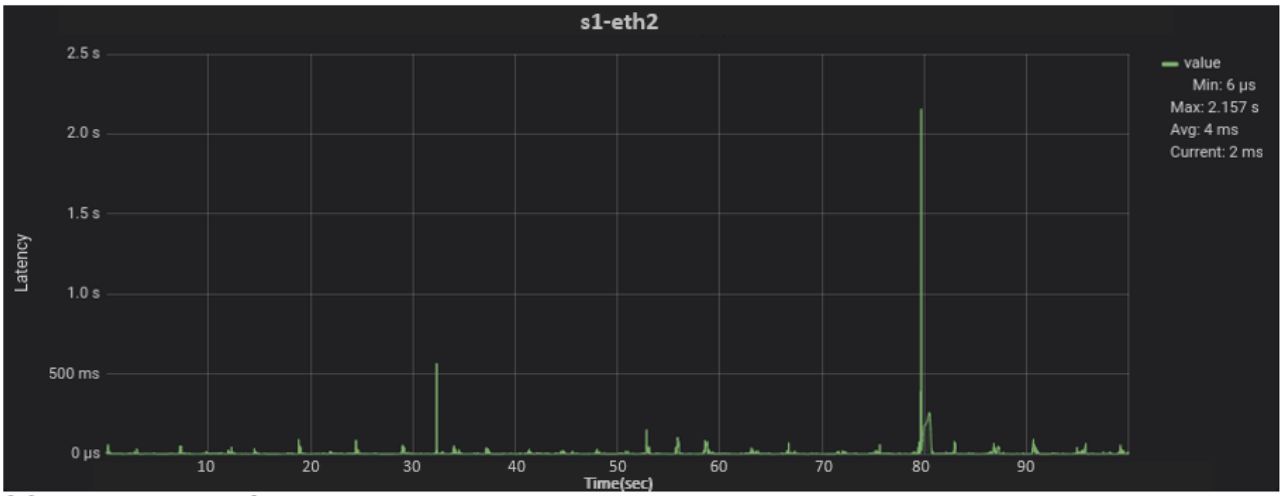


(b) Latency on Switch 2

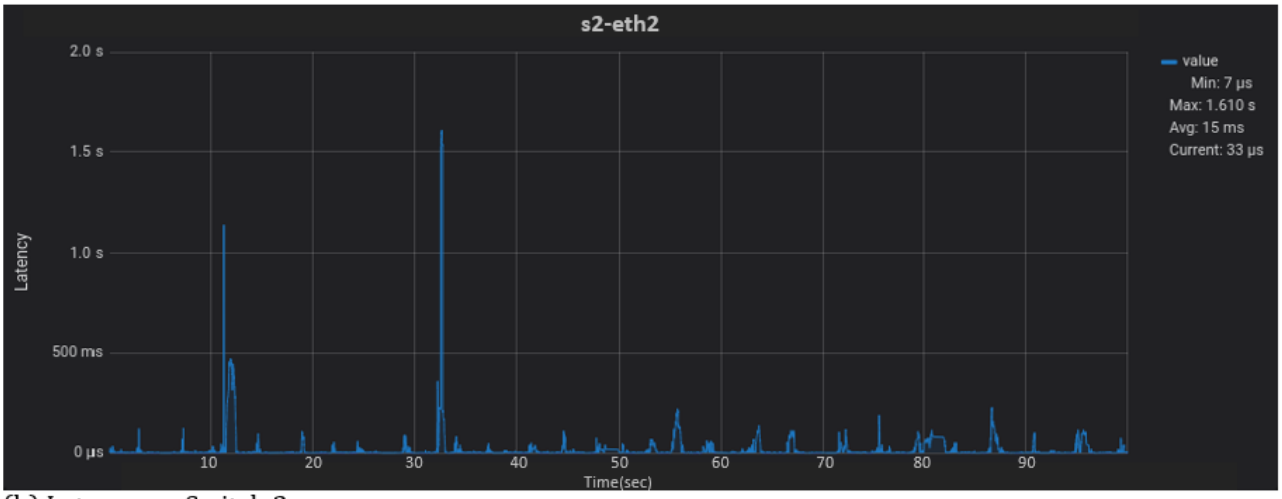
Figure 5.7: Switch 1 & Switch 2 Latency during traffic exchange between h11 and h22

Latency is normally said to be the time it takes to send packets from a host in one location to another host in a different location. However, latency can also be a measure of time packets spent propagating an ethernet switch.

Figure 5.7 show that packets spend queueing on switch 1 and switch 2 of the network during traffic exchange between the high priority hosts h11 and h22. The results show that even though there are packet queue spikes of over 7 milliseconds, both the switches on average perform very well with average latencies of 117 microseconds and 187 microseconds. The performance of the switches is



(1) Latency on Switch 1



(b) Latency on Switch 2

Figure 5.8: Switch 1 & Switch 2 Latency during traffic exchange between h1 and h2

due to fact that high priority traffic class is given preference and therefore, less queueing time.

The latency of switches 1 and 2 during traffic exchange between hosts h1 and h2 is shown in Figure 5.8. The switches experiences shorter maximum packet queues compared to high priority traffic, but longer packet queue delays of 4 milliseconds and 15 milliseconds. The reason for longer latencies in this traffic class is clear, and this because packets are kept in queues a little longer to allow high priority packets to be transmitted.

The results of this section indicate how P4 program can be used to define packet process packets and how network controllers such as P4Runtime can be utilized to manage traffic flow and provide the least congestion solution for better real-time applications performance. These capabilities are useful in devices such as car control systems to mitigate car crushes or accidents by taking control prior to human reaction.

6. Conclusions

The demand for TSN is due to its ability to perform real-time critical communication by forwarding industrial networks critical processes data. Real-time applications would often run in a separate network from other application; however, the quickly evolving innovation requires these applications to not only have real-time capabilities, but to have other functionalities. TSN allows different data streams to share the same network net yet ensuring that real-time applications data is prioritized and arrives on time.

This research project presented the context of Time-Sensitive Networking by carrying out a TSN switch and deployed a TSN network consisting of various switches and a network controller used to perform experiments. Furthermore, basic IP switching and tunneling were implemented on the switch to allow the switches to perform basic packet forwarding, and as a necessity for the network to handle different streams of data.

The results showed that when the operates as an ethernet communication, data on the network is handled as best-effort and all applications will share the network resources equally. On the other hand, the results show that TSN can be used to control communication between applications by giving real-time applications priority and allow them to utilise network resources such as bandwidth to ensure guaranteed reliability. The results also show that applications with access to these network resources will have less packet loss and less jitter.

Another observation is that, while real-time traffic is allocation most of the bandwidth, non-real-time traffic (low priority) are stripped off all the resources and allocated next to nothing. This is because low priority traffic is allocated the least priority of 0 and high priority host are allocated the maximum priority of 7 in the traffic scheduling egress port gating mechanism. The operation can always be altered to suit operation of any network and priority can be allocated in any way possible to give best achievable network performance.

The implementation of the made of P4 programming and the switch of choice is a v1model Bmv2. P4 is a high-level language and unlike the low-level languages, it can program the vendor specific switch interfaces. The ability of P4 to protocol-independent was used as an advantage to configure the switches packet parsers by defining a set of match-action tables and performing TSN operational specifications.

The concept of SDN of decoupling control plane and data plane and centralizing network communication was adopted and implemented using P4Runtime. P4Runtime behaved as controller that applied the rules defined by the P4 program written to switches processing pipelines and applied them tables by installing the control plane rules that performs tasks such as the match-action, add, remove and update of tables.

7. Recommendations

The following recommendations are drawn based on the conclusion made, test results obtained on the experiments, knowledge, and experience developed during the period of conducting this research project.

7.1 Possible Improvements

The TSN network implemented in this research project allocates and gate only 0 and gate 7 of the BMv2 switch to perform IEEE 802.1Qbv (Traffic Scheduling) and the results show that even though priority is achieved, low priority traffic is stripped of almost all the resource. The conclusion indicated that this can altered to give possible various network performance. Since the switch consist of eight time-aware gates, eight different traffic types can be assigned in a switch. The different types of traffic allocation in a network can be seen found in a modern cars network, the network would consist of traffic types such as audio/video, sensors, cameras, GPS, and control systems.

Furthermore, the experiments performed in this project focuses on between two traffic classes, however, real life systems such as car system mentioned above, and industrial factories consists of bigger larger traffic classes. The experiment can be to utilize the all the available switch gates and perform experiments to see how different classes and network size influence the performance of the network.

7.2 Future Work

This research projects performs the design and implementation of TSN on a programmable software switch and deployed virtual TSN network. Future work would be to extend these TSN specifications, SDN and P4 program to be performed on programmable hardware devices such as FPGA and ASICs.

The research would be conducted on how these switches can be programmed to be used in an Industrial Control System (ICS) environment to process information from devices such as Programmable Logic Systems (PLC) and other devices, synchronise relevant communicating devices and control the operation of the factories Wide Area Network (WAN) traffic.

8. List of References

[1] V. Dadwal, "Adopting Time-Sensitive Networking (TSN) for Automation Systems", Intel, 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/adopting-time-sensitive-networking-tsn-for-automation-systems-0.html. [Accessed: 03- Nov- 2020].

[2] T. Häckel, P. Meyer, F. Korf and T. Schmidt, "SDN4CoRE: A Simulation Model for Software-Defined Networking for Communication over Real-Time Ethernet", Semanticscholar.org, 2019. [Online]. Available: https://www.semanticscholar.org/paper/SDN4CoRE%3A-A-Simulation-Model-for-Software-Defined-H%C3%A4ckel-Meyer/006ebec79f4bf8d139d57d365d042550c75ed327. [Accessed: 03- Nov- 2020].

[3] M. Muminovic and H. Suljic, "Performance Study and Analysis of Time Sensitive Networking", DIVA, 2019. [Online]. Available: http://mdh.diva-portal.org/smash/record.jsf?pid=diva2%3A1324903&dswid=-5209. [Accessed: 03- Nov- 2020].

[4] A. Bergström, "Automatic generation of network configuration in simulated time sensitive networking (TSN) applications", DIVA, 2020. [Online]. Available: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1438082&dswid=4674. [Accessed: 03- Nov- 2020].

[5] H. Laine, "Simulating Time-Sensitive Networking", Semanticscholar.org, 2015. [Online]. Available: https://www.semanticscholar.org/paper/Simulating-Time-Sensitive-Networking-Laine/f3b176b6fc7e5ba8a4e6053ee6c3ddaa928dff56. [Accessed: 03- Nov- 2020].

[6] D. Klein and M. Jerschel, "An OpenFlow extension for the OMNeT++ INET framework | Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques", Dl.acm.org, 2013. [Online]. Available: https://dl.acm.org/doi/10.5555/2512734.2512780. [Accessed: 03- Nov- 2020].

[7] T. Hackel, P. Meyer, F. Korf and T. Schmid, "Software-Defined Networks Supporting Time-Sensitive In-Vehicular Communication", Ieeexplore-ieee-org.ezproxy.uct.ac.za, 2019. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.uct.ac.za/document/8746473. [Accessed: 03- Nov- 2020].

[8] R. Kundel, F. Siegmund, J. Blendin, A. Rizk and B. Koldehofe, "P4STA: High Performance Packet Timestamping with Programmable Packet Processors", Ieeexplore-ieee-org.ezproxy.uct.ac.za, 2020. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.uct.ac.za/document/9110290. [Accessed: 03- Nov- 2020].

[9] P. Varis and T. Leyrer, "Time-sensitive networking for industrial automation", Ti.com, 2020. [Online]. Available: https://www.ti.com/lit/wp/spry316a/spry316a.pdf?ts=1604415041196&ref_url=https%253A%252F%252Fwww.google.de%252F. [Accessed: 03- Nov- 2020].

[10] P. Bosshart et al., "P4: Programming Protocol-Independent Packet Processors", Sigcomm.org, 2014. [Online]. Available: https://www.sigcomm.org/sites/default/files/ccr/papers/2014/July/0000000-0000004.pdf. [Accessed: 03- Nov- 2020].

[11] J. Vestin, "SDN-Enabled Resiliency, Monitoring and Control in Computer Networks", DIVA, 2020. [Online]. Available: http://kau.diva-portal.org/smash/record.jsf?pid=diva2%3A1377732&dswid=4674. [Accessed: 03- Nov- 2020].

[12] P. Bosshart et al., "Programming Protocol-Independent Packet Processors", arXiv.org, 2013. [Online]. Available: https://arxiv.org/abs/1312.1719. [Accessed: 03- Nov- 2020].

[13] W. Quan, W. Fu, J. Yan and Z. Sun, "OpenTSN: an open-source project for time-sensitive networking system development", researchgate, 2020. [Online]. Available: https://www.researchgate.net/publication/343469636_OpenTSN_an_open-source_project_for_time-sensitive_networking_system_development. [Accessed: 03- Nov- 2020].

[14] S. Brooks and E. Uludag, "Time-Sensitive Networking: From Theory to Implementation in Industrial Automation", Intel.com, 2019. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01279-time-sensitive-networking-from-theory-to-implementation-in-industrial-automation.pdf. [Accessed: 03- Nov- 2020].

[15] M. Budiu, "Programming networks with P4 - VMware Research", VMware Research, 2017. [Online]. Available: https://blogs.vmware.com/research/2017/04/07/programming-networks-p4/#:~:text=P4%20is%20a%20domain%2Dspecific,routers%20or%20network%20function%20appliances. [Accessed: 03- Nov- 2020].

[16] S. Byrd, "P4: Driving Innovation in Server-Based Networking - PDF Free Download", Docplayer.net, 2017. [Online]. Available: https://docplayer.net/49066874-P4-driving-innovation-in-server-based-networking.html. [Accessed: 03- Nov- 2020].

[17] R. Hummen, "What is TSN? A Look at Its Role in Future Ethernet Networks", Belden.com, 2017. [Online]. Available: https://www.belden.com/blog/industrial-ethernet/what-is-tsn-a-look-at-its-role-in-future-ethernet-networks. [Accessed: 03- Nov- 2020].

[18] Z. Cao, H. Su, Q. Yang, J. Shen, M. Wen and C. Zhang, "P4 to FPGA-A Fast Approach for Generating Efficient Network Processors", Ieeexplore-ieee-org.ezproxy.uct.ac.za, 2020. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.uct.ac.za/document/8976091. [Accessed: 03- Nov- 2020].

[19] H. Wang et al., "P4FPGA | Proceedings of the Symposium on SDN Research", Dl.acm.org, 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3050220.3050234. [Accessed: 03- Nov- 2020].

[20] D. Kreutz, F. Ramos, P. Veríssimo, C. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", Ieeexplore-ieee-org.ezproxy.uct.ac.za, 2015. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.uct.ac.za/document/6994333. [Accessed: 03- Nov- 2020].

[21] L. Silva, P. Pedreiras, P. Fonseca and L. Almeida, "On the adequacy of SDN and TSN for Industry 4.0", Ieeexplore-ieee-org.ezproxy.uct.ac.za, 2019. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.uct.ac.za/stamp/stamp.jsp?tp=&arnumber=8759277. [Accessed: 03- Nov- 2020].

[22] F. Geyer and M. Winkel, "Towards Embedded Packet Processing Devices for Rapid Prototyping of Avionic Applications", Semanticscholar.org, 2018. [Online]. Available:

https://www.semanticscholar.org/paper/Towards-Embedded-Packet-Processing-Devices-for-of-Geyer-Winkel/d8bde42b687b289d303c7089049e623fc2385917. [Accessed: 03- Nov- 2020].

[23] I. Safuilin and A. Geissler, "Deterministic ethernet and IEEE TSN for Aerospace", Modern-avionics.ru, 2019. [Online]. Available: http://www.modern-avionics.ru/Files/22-TTTech-Safiulin-29.08.2019.pdf. [Accessed: 03- Nov- 2020].

[24] B. Vinnakota, M. Pham and J. Gao, "P4: A Hands-on Introduction", Nfvsdn2016.ieee-nfvsdn.org, 2016. [Online]. Available: http://nfvsdn2016.ieee-nfvsdn.org/files/2016/09/p4_nfv_conference_tutorial_proposal.pdf. [Accessed: 03- Nov- 2020].

[25] S. Krishnan, "P4 and After", Kaloom.com, 2020. [Online]. Available: https://www.kaloom.com/blog/p4-and-after. [Accessed: 03- Nov- 2020].

[26] G. Kumar, K. Katsalis and P. Papadimitriou, "Coupling Source Routing with Time-Sensitive Networking", Ieeexplore-ieee-org.ezproxy.uct.ac.za, 2020. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.uct.ac.za/document/9142793. [Accessed: 03- Nov- 2020].

[27] N. Nayak, F. Dürr and K. Rothermel, "Routing algorithms for IEEE802.1Qbv networks", Semanticscholar.org, 2018. [Online]. Available: https://www.semanticscholar.org/paper/Routing-algorithms-for-IEEE802.1Qbv-networks-Nayak-D%C3%BCrr/377f7e79f88bdad6d42ed414e35d9edcc29ff5d9. [Accessed: 03- Nov- 2020].

[28] S. Mehner, "[ovs-discuss] Extending OVS with TSN functionalities (IEEE 802.1Qbv)", Mail-archive.com, 2020. [Online]. Available: https://www.mail-archive.com/ovs-discuss@openvswitch.org/msg07304.html. [Accessed: 03- Nov- 2020].

[29] J. Sanchez-Palencia, "The Road Towards a Linux TSN Infrastructure", Elinux.org, 2018. [Online]. Available: https://elinux.org/images/5/56/ELC-2018-USA-TSNonLinux.pdf. [Accessed: 03- Nov- 2020].

[30] A. Nasrallah et al., "Ultra-Low Latency (ULL) Networks: A Comprehensive Survey Covering the IEEE TSN Standard and Related ULL Research", Semanticscholar.org, 2018. [Online]. Available: https://www.semanticscholar.org/paper/Ultra-Low-Latency-(ULL)-Networks%3A-A-Comprehensive-Nasrallah-Thyagaturu/4bf7a789a0e4c432e97131bc2e98a4052f5c5455. [Accessed: 03- Nov- 2020].

[31] M. B¨ohm, J. Ohms, M. Kumar, O. Gebauer and D. Wermser, "Dynamic Real-Time Stream Reservation for IEEE 802.1 Time-Sensitive Networks with OpenFlow", Opendata.uni-halle.de, 2020. [Online]. Available: https://opendata.uni-halle.de/bitstream/1981185920/32925/1/1_2_Boehem.pdf. [Accessed: 04- Nov- 2020].

[32] K. Lee, S. Lee and M. Lee, "Worst Case Communication Delay of Real-Time Industrial Switched Ethernet With Multiple Levels", Ieeexplore-ieee-org.ezproxy.uct.ac.za, 2020. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.uct.ac.za/document/1705659. [Accessed: 04- Nov- 2020].

[33] T. Brand, "Time Sensitive Networks: Real-Time Ethernet | Analog Devices", Analog.com, 2020. [Online]. Available: https://www.analog.com/en/technical-articles/a87000-time-sensitive-networks-real-time-ethernet.html. [Accessed: 04- Nov- 2020].

[34] J. Mwangama, "2020 EEE 4121F Module B - sec 04 - Programmable Network Data Planes", Google Docs, 2020. [Online]. Available: https://docs.google.com/presentation/d/1IYV_5mDQ4XBUKF2_ZS-fhvxiDjhSev3OpmIms_b4V58/edit. [Accessed: 04- Nov- 2020].

[35] L. Consortium, "P4 Language Tutorial", P4.org, 2017. [Online]. Available: https://p4.org/assets/P4_tutorial_01_basics.gslide.pdf. [Accessed: 04- Nov- 2020].

[36] B. Turkovic, F. Kuipers, N. Adrichem and K. Langendoen, "Fast network congestion detection and avoidance using P4 | Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies", Dl.acm.org, 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3229574.3229581. [Accessed: 04- Nov- 2020].

[37] A. Armstrong, "Precision Time Protocol on Linux ~ Introduction to linuxptp", Silo.tips, 2017. [Online]. Available: https://silo.tips/download/precision-time-protocol-on-linux-introduction-to-linuxptp. [Accessed: 04- Nov- 2020].

[38] K. Tokmakov, "Advanced data centre traffic management on programmable ethernet switch infrastructure", Oparu.uni-ulm.de, 2019. [Online]. Available: https://oparu.uni-ulm.de/xmlui/handle/123456789/16443. [Accessed: 04- Nov- 2020].

[39] Mininet Team, "Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet", Mininet.org. [Online]. Available: http://mininet.org/. [Accessed: 04- Nov- 2020].

[40] P4 language consortium, "p4language", GitHub. [Online]. Available: https://github.com/p4lang. [Accessed: 04- Nov- 2020].

[41] e. costa, "nsg-ethz/p4-learning", GitHub, 2020. [Online]. Available: https://github.com/nsg-ethz/p4-learning/tree/master/examples. [Accessed: 04- Nov- 2020].

[42] "Time-Sensitive Networking (TSN) Task Group |", 1.ieee802.org. [Online]. Available: https://1.ieee802.org/tsn/. [Accessed: 04- Nov- 2020].

[43] V. GUEANT, "iPerf - The TCP, UDP and SCTP network bandwidth measurement tool", Iperf.fr. [Online]. Available: https://iperf.fr/. [Accessed: 04- Nov- 2020].

[44] "gnuplot homepage", Gnuplot.info. [Online]. Available: http://www.gnuplot.info/. [Accessed: 04- Nov- 2020].

9. Appendices

9.1 Appendix A

P4 Source Code

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

const bit<8>  TCP_PROTOCOL = 0x06;
const bit<16> TYPE_MYTUNNEL = 0x1212;
const bit<16> TYPE_IPV4 = 0x800;

/*************************************************************************
********************** H E A D E R S  ***********************************
*************************************************************************/

typedef bit<9>  egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

header ethernet_t {
   macAddr_t dstAddr;
   macAddr_t srcAddr;
   bit<16>   etherType;
}

header myTunnel_t {
   bit<16> proto_id;
   bit<16> dst_id;
}

header ipv4_t {
   bit<4>    version;
   bit<4>    ihl;
   bit<8>    diffserv;
   bit<16>   totalLen;
   bit<16>   identification;
   bit<3>    flags;
   bit<13>   fragOffset;
   bit<8>    ttl;
   bit<8>    protocol;
   bit<16>   hdrChecksum;
   ip4Addr_t srcAddr;
```

```
    ip4Addr_t dstAddr;
}

struct metadata {
}

struct headers {
    ethernet_t   ethernet;
    myTunnel_t   myTunnel;
    ipv4_t       ipv4;
}

/*************************************************************************
********************** P A R S E R  ***********************************
*************************************************************************/

parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_MYTUNNEL: parse_myTunnel;
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_myTunnel {
        packet.extract(hdr.myTunnel);
        transition select(hdr.myTunnel.proto_id) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
    }
}

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }

}
```

```
/*************************************************************************
************  C H E C K S U M   V E R I F I C A T I O N  *************
*************************************************************************/

control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
   apply { }
}


/*************************************************************************
**************  I N G R E S S   P R O C E S S I N G  *******************
*************************************************************************/

control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {

   counter(MAX_TUNNEL_ID, CounterType.packets_and_bytes) ingressTunnelCounter;
   counter(MAX_TUNNEL_ID, CounterType.packets_and_bytes) egressTunnelCounter;

   action drop() {
      mark_to_drop(standard_metadata);
   }

   action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
      standard_metadata.egress_spec = port;
      hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
      hdr.ethernet.dstAddr = dstAddr;
      hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
   }

   action myTunnel_ingress(bit<16> dst_id) {
      hdr.myTunnel.setValid();
      hdr.myTunnel.dst_id = dst_id;
      hdr.myTunnel.proto_id = hdr.ethernet.etherType;
      hdr.ethernet.etherType = TYPE_MYTUNNEL;
      ingressTunnelCounter.count((bit<32>) hdr.myTunnel.dst_id);
   }

   action myTunnel_forward(egressSpec_t port) {
      standard_metadata.egress_spec = port;
   }

   action myTunnel_egress(macAddr_t dstAddr, egressSpec_t port) {
      standard_metadata.egress_spec = port;
      hdr.ethernet.dstAddr = dstAddr;
      hdr.ethernet.etherType = hdr.myTunnel.proto_id;
```

```
    hdr.myTunnel.setInvalid();
    egressTunnelCounter.count((bit<32>) hdr.myTunnel.dst_id);
}

table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        myTunnel_ingress;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}

table myTunnel_exact {
    key = {
        hdr.myTunnel.dst_id: exact;
    }
    actions = {
        myTunnel_forward;
        myTunnel_egress;
        drop;
    }
    size = 1024;
    default_action = drop();
}

apply {
    if (hdr.ipv4.isValid() && !hdr.myTunnel.isValid()) {
        // Process only non-tunneled IPv4 packets.
        ipv4_lpm.apply();
    }

    if (hdr.myTunnel.isValid()) {
        // Process all tunneled packets.
        myTunnel_exact.apply();
    }
apply {

    if (hdr.ipv4.isValid()){
        ipv4_lpm.apply();
        if (hdr.ipv4.srcAddr == 0x0a000101){
            standard_metadata.priority = (bit<3>)0;
        }
        else if (hdr.ipv4.srcAddr == 0x0a00010b){
```

```
            standard_metadata.priority = (bit<3>)7;
        }
      }
    }
}

/*************************************************************************
**************** E G R E S S  P R O C E S S I N G  ******************
*************************************************************************/

control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    apply { }
}

/*************************************************************************
*************  C H E C K S U M  C O M P U T A T I O N  **************
*************************************************************************/

control MyComputeChecksum(inout headers hdr, inout metadata meta) {
    apply {
        /* TODO: replace tos with diffserve and ecn */
            update_checksum(
                hdr.ipv4.isValid(),
            { hdr.ipv4.version,
              hdr.ipv4.ihl,
              hdr.ipv4.diffserv,
              hdr.ipv4.totalLen,
              hdr.ipv4.identification,
              hdr.ipv4.flags,
              hdr.ipv4.fragOffset,
              hdr.ipv4.ttl,
              hdr.ipv4.protocol,
              hdr.ipv4.srcAddr,
              hdr.ipv4.dstAddr },
            hdr.ipv4.hdrChecksum,
            HashAlgorithm.csum16);
    }
}

/*************************************************************************
***********************  D E P A R S E R  ********************************
*************************************************************************/

control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.myTunnel);
```

```
        packet.emit(hdr.ipv4);
    }
}

/*************************************************************************
**********************  S W I T C H  ****************************
*************************************************************************/

V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

9.2 Appendix B

Source Code Link: https://github.com/mkwjoh004uct/A-Time-Sensitive-network-Switch-Implementation-using-P4

9.3 Appendix C

Extras

C.1 TSN Network Implementation Bandwidth, Packet Loss and Jitter Results Data

Traffic Exchange between h1&h2 with priority disabled

```
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 24] local 10.0.2.2 port 5001 connected with 10.0.1.1 port 39523
[ ID] Interval Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[24] 01.0 sec  1.15 MBytes  9.68 Mbits/sec  0.459 ms  0/ 823  0 %
[24] 02.0 sec  1.12 MBytes  9.38 Mbits/sec  0.937 ms  0/ 798  0 %
[24] 03.0 sec  1.13 MBytes  9.49 Mbits/sec  0.288 ms  4/ 811  0.49 %
[24] 04.0 sec  1.12 MBytes  9.43 Mbits/sec  0.339 ms  1/ 803  0.12 %
[24] 05.0 sec  1.12 MBytes  9.42 Mbits/sec  0.549 ms  0/ 801  0 %
[24] 06.0 sec  1.16 MBytes  9.69 Mbits/sec  0.366 ms  0/ 824  0 %
[24] 07.0 sec  1.16 MBytes  9.70 Mbits/sec  0.636 ms  19/ 844  2.3 %
[24] 08.0 sec  1.10 MBytes  9.23 Mbits/sec  1.421 ms  32/ 817  3.9 %
[24] 09.0 sec  1.15 MBytes  9.68 Mbits/sec  0.407 ms  60/ 883  6.8 %
[24] 10.0 sec  1.13 MBytes  9.47 Mbits/sec  0.792 ms  45/ 850  5.3 %
[24] 11.0 sec  1.16 MBytes  9.69 Mbits/sec  0.643 ms  28/ 852  3.3 %
[24] 12.0 sec  1.11 MBytes  9.29 Mbits/sec  0.224 ms  29/ 819  3.5 %
[24] 13.0 sec  1.12 MBytes  9.40 Mbits/sec  0.869 ms  67/ 866  7.7 %
[24] 14.0 sec  1.15 MBytes  9.68 Mbits/sec  0.473 ms  43/ 866  5 %
[24] 15.0 sec  1.15 MBytes  9.63 Mbits/sec  0.385 ms  32/ 851  3.8 %
[24] 16.0 sec  689 KBytes   5.64 Mbits/sec  3.504 ms  368/ 848  43 %
[24] 17.0 sec  579 KBytes   4.74 Mbits/sec  2.554 ms  435/ 838  52 %
[24] 18.0 sec  591 KBytes   4.85 Mbits/sec  2.080 ms  428/ 840  51 %
[24] 19.0 sec  583 KBytes   4.77 Mbits/sec  1.920 ms  443/ 849  52 %
[24] 20.0 sec  586 KBytes   4.80 Mbits/sec  3.046 ms  465/ 873  53 %
[24] 21.0 sec  583 KBytes   4.77 Mbits/sec  2.489 ms  436/ 842  52 %
[24] 22.0 sec  584 KBytes   4.79 Mbits/sec  1.930 ms  430/ 837  51 %
[24] 23.0 sec  579 KBytes   4.74 Mbits/sec  2.850 ms  463/ 866  53 %
[24] 24.0 sec  560 KBytes   4.59 Mbits/sec  3.226 ms  407/ 797  51 %
[24] 25.0 sec  537 KBytes   4.40 Mbits/sec  2.658 ms  525/ 899  58 %
[24] 26.0 sec  587 KBytes   4.81 Mbits/sec  2.179 ms  439/ 848  52 %
[24] 27.0 sec  577 KBytes   4.73 Mbits/sec  2.018 ms  441/ 843  52 %
[24] 28.0 sec  587 KBytes   4.81 Mbits/sec  3.487 ms  461/ 870  53 %
[24] 29.0 sec  568 KBytes   4.66 Mbits/sec  3.271 ms  447/ 843  53 %
[24] 30.0 sec  577 KBytes   4.73 Mbits/sec  3.091 ms  440/ 842  52 %
[24] 31.0 sec  583 KBytes   4.77 Mbits/sec  2.636 ms  451/ 857  53 %
[24] 32.0 sec  590 KBytes   4.83 Mbits/sec  2.235 ms  436/ 847  51 %
[24] 33.0 sec  586 KBytes   4.80 Mbits/sec  1.923 ms  436/ 844  52 %
[24] 34.0 sec  579 KBytes   4.74 Mbits/sec  1.818 ms  441/ 844  52 %
[24] 35.0 sec  590 KBytes   4.83 Mbits/sec  2.930 ms  463/ 874  53 %
[24] 36.0 sec  580 KBytes   4.75 Mbits/sec  2.674 ms  443/ 847  52 %
[24] 37.0 sec  587 KBytes   4.81 Mbits/sec  2.141 ms  432/ 841  51 %
[24] 38.0 sec  583 KBytes   4.77 Mbits/sec  2.771 ms  456/ 862  53 %
[24] 39.0 sec  581 KBytes   4.76 Mbits/sec  2.552 ms  430/ 835  51 %
[24] 40.0 sec  577 KBytes   4.73 Mbits/sec  1.963 ms  445/ 847  53 %
[24] 41.0 sec  580 KBytes   4.75 Mbits/sec  2.493 ms  458/ 862  53 %
[24] 42.0 sec  580 KBytes   4.75 Mbits/sec  1.937 ms  434/ 838  52 %
[24] 43.0 sec  587 KBytes   4.81 Mbits/sec  2.785 ms  458/ 867  53 %
[24] 44.0 sec  581 KBytes   4.76 Mbits/sec  2.040 ms  436/ 841  52 %
[24] 45.0 sec  571 KBytes   4.68 Mbits/sec  2.218 ms  451/ 849  53 %
[24] 46.0 sec  877 KBytes   7.19 Mbits/sec  1.584 ms  125/ 736  17 %
[24] 47.0 sec  663 KBytes   5.43 Mbits/sec  1.405 ms  460/ 922  50 %
[24] 48.0 sec  1.02 MBytes  8.53 Mbits/sec  3.555 ms  125/ 850  15 %
[24] 49.0 sec  936 KBytes   7.67 Mbits/sec  0.971 ms  256/ 908  28 %
[24] 50.0 sec  1.05 MBytes  8.78 Mbits/sec  1.847 ms  68/ 815  8.3 %
[24] 51.0 sec  1.13 MBytes  9.49 Mbits/sec  0.444 ms  82/ 889  9.2 %
[24] 52.0 sec  1.14 MBytes  9.55 Mbits/sec  0.304 ms  37/ 849  4.4 %
[24] 53.0 sec  1.06 MBytes  8.87 Mbits/sec  0.832 ms  95/ 849  11 %
[24] 54.0 sec  1.04 MBytes  8.73 Mbits/sec  0.982 ms  84/ 826  10 %
[24] 55.0 sec  1.13 MBytes  9.47 Mbits/sec  0.514 ms  71/ 876  8.1 %
[24] 56.0 sec  1.15 MBytes  9.62 Mbits/sec  0.371 ms  27/ 845  3.2 %
[24] 57.0 sec  1.15 MBytes  9.62 Mbits/sec  0.310 ms  36/ 854  4.2 %
[24] 58.0 sec  1.15 MBytes  9.64 Mbits/sec  0.518 ms  31/ 851  3.6 %
[24] 59.0 sec  1.15 MBytes  9.65 Mbits/sec  0.576 ms  29/ 850  3.4 %
[24] 60.0 sec  1.15 MBytes  9.63 Mbits/sec  0.508 ms  32/ 851  3.8 %
```

Traffic Exchange between h1&h2 with priority enabled

```
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 24] local 10.0.2.2 port 5001 connected with 10.0.1.1 port 52315
[ ID] Interval  Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[24]  01.0 sec  1.14 MBytes  9.58 Mbits/sec  0.250 ms  0/  815  0 %
[24]  02.0 sec  1.09 MBytes  9.18 Mbits/sec  0.234 ms  1/  782  0.13 %
[24]  03.0 sec  1.15 MBytes  9.63 Mbits/sec  0.346 ms  0/  819  0 %
[24]  04.0 sec  1.15 MBytes  9.61 Mbits/sec  0.354 ms  0/  817  0 %
[24]  05.0 sec  1.04 MBytes  8.71 Mbits/sec  1.220 ms  2/  743  0.27 %
[24]  06.0 sec  1.07 MBytes  9.01 Mbits/sec  1.359 ms  74/ 840  8.8 %
[24]  07.0 sec  1.03 MBytes  8.67 Mbits/sec  3.342 ms  83/ 820  10 %
[24]  08.0 sec  1.11 MBytes  9.35 Mbits/sec  0.597 ms  123/ 918  13 %
[24]  09.0 sec  1.13 MBytes  9.49 Mbits/sec  0.466 ms  42/  849  4.9 %
[24]  10.0 sec  1.15 MBytes  9.67 Mbits/sec  0.388 ms  129/ 951  14 %
[24]  11.0 sec  1.15 MBytes  9.67 Mbits/sec  0.344 ms  0/  822  0 %
[24]  12.0 sec  1.14 MBytes  9.60 Mbits/sec  0.558 ms  0/  816  0 %
[24]  13.0 sec  1.14 MBytes  9.58 Mbits/sec  0.333 ms  0/  815  0 %
[24]  14.0 sec  1.14 MBytes  9.58 Mbits/sec  0.408 ms  30/  845  3.6 %
[24]  15.0 sec  1.14 MBytes  9.55 Mbits/sec  0.293 ms  39/  851  4.6 %
[27]  16.0 sec  201 KBytes   0.65 Mbits/sec  3.093 ms  30/  170  88 %
[27]  17.0 sec  201 KBytes   0.65 Mbits/sec  2.463 ms  23/  163  84 %
[27]  18.0 sec  201 KBytes   0.65 Mbits/sec  2.276 ms  29/  169  87 %
[27]  19.0 sec  189 KBytes   0.55 Mbits/sec  4.699 ms  38/  170  92 %
[27]  20.0 sec  182 KBytes   0.49 Mbits/sec  2.581 ms  41/  168  94 %
[27]  21.0 sec  185 KBytes   0.52 Mbits/sec  5.567 ms  45/  174  96 %
[27]  22.0 sec  195 KBytes   0.60 Mbits/sec  3.588 ms  34/  170  90 %
[27]  23.0 sec  192 KBytes   0.58 Mbits/sec  2.668 ms  36/  170  91 %
[27]  24.0 sec  197 KBytes   0.61 Mbits/sec  2.237 ms  32/  169  89 %
[27]  25.0 sec  200 KBytes   0.63 Mbits/sec  3.945 ms  33/  172  89 %
[27]  26.0 sec  194 KBytes   0.59 Mbits/sec  2.522 ms  33/  168  90 %
[27]  27.0 sec  198 KBytes   0.62 Mbits/sec  2.629 ms  34/  172  90 %
[27]  28.0 sec  195 KBytes   0.60 Mbits/sec  2.854 ms  32/  168  89 %
[27]  29.0 sec  201 KBytes   0.65 Mbits/sec  2.543 ms  30/  170  88 %
[27]  30.0 sec  192 KBytes   0.58 Mbits/sec  2.823 ms  32/  166  89 %
[27]  31.0 sec  185 KBytes   0.52 Mbits/sec  5.075 ms  46/  175  96 %
[27]  32.0 sec  201 KBytes   0.65 Mbits/sec  3.093 ms  30/  170  88 %
[27]  33.0 sec  194 KBytes   0.59 Mbits/sec  3.261 ms  36/  171  91 %
[27]  34.0 sec  200 KBytes   0.63 Mbits/sec  2.758 ms  30/  169  88 %
[27]  35.0 sec  178 KBytes   0.46 Mbits/sec  3.292 ms  38/  162  93 %
[27]  36.0 sec  201 KBytes   0.65 Mbits/sec  2.764 ms  38/  178  91 %
[27]  37.0 sec  195 KBytes   0.60 Mbits/sec  2.564 ms  34/  170  90 %
[27]  38.0 sec  202 KBytes   0.66 Mbits/sec  2.938 ms  31/  172  88 %
[27]  39.0 sec  200 KBytes   0.63 Mbits/sec  2.297 ms  30/  169  88 %
[27]  40.0 sec  195 KBytes   0.60 Mbits/sec  2.434 ms  31/  167  89 %
[27]  41.0 sec  178 KBytes   0.46 Mbits/sec  2.665 ms  49/  173  98 %
[27]  42.0 sec  198 KBytes   0.62 Mbits/sec  2.419 ms  31/  169  88 %
[27]  43.0 sec  201 KBytes   0.65 Mbits/sec  2.776 ms  31/  171  88 %
[27]  44.0 sec  200 KBytes   0.63 Mbits/sec  2.545 ms  30/  169  88 %
[27]  45.0 sec  200 KBytes   0.63 Mbits/sec  2.756 ms  33/  172  89 %
[28]  46.0 sec  1.15 MBytes  9.61 Mbits/sec  0.604 ms  36/  853  4.2 %
[28]  47.0 sec  1.12 MBytes  9.36 Mbits/sec  0.443 ms  56/  852  6.6 %
[28]  48.0 sec  1.13 MBytes  9.51 Mbits/sec  0.805 ms  32/  841  3.8 %
[28]  49.0 sec  1.13 MBytes  9.44 Mbits/sec  0.540 ms  63/  866  7.3 %
[28]  50.0 sec  1.12 MBytes  9.37 Mbits/sec  0.924 ms  42/  839  5 %
[28]  51.0 sec  1.11 MBytes  9.34 Mbits/sec  1.076 ms  50/  844  5.9 %
[28]  52.0 sec  1.12 MBytes  9.40 Mbits/sec  1.402 ms  53/  852  6.2 %
[28]  53.0 sec  1.16 MBytes  9.71 Mbits/sec  0.357 ms  40/  866  4.6 %
[28]  54.0 sec  1.15 MBytes  9.61 Mbits/sec  0.491 ms  30/  847  3.5 %
[28]  55.0 sec  1.16 MBytes  9.69 Mbits/sec  0.491 ms  30/  854  3.5 %
[28]  56.0 sec  1.12 MBytes  9.36 Mbits/sec  0.443 ms  56/  852  6.6 %
[28]  57.0 sec  1.15 MBytes  9.67 Mbits/sec  0.290 ms  25/  847  3 %
[28]  58.0 sec  1.16 MBytes  9.69 Mbits/sec  0.481 ms  25/  849  2.9 %
[28]  59.0 sec  1.12 MBytes  9.37 Mbits/sec  0.605 ms  53/  850  6.2 %
[28]  60.0 sec  1.15 MBytes  9.61 Mbits/sec  0.604 ms  36/  853  4.2 %
```

Traffic Exchange between h11&h22 with priority disabled

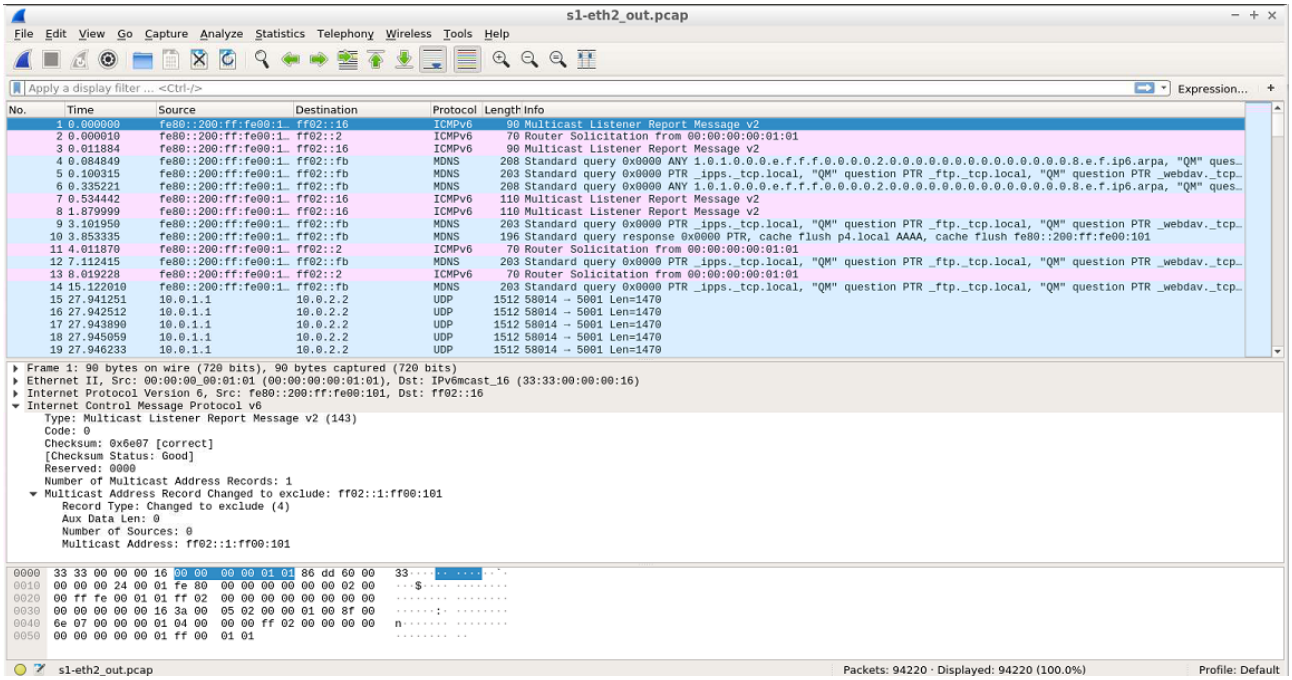
```
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 27] local 10.0.2.22 port 5001 connected with 10.0.1.11 port 54879
[ ID] Interval  Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[27] 01.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 02.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 03.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 04.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 05.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 06.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 07.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 08.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 09.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 10.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 11.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 12.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 13.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 14.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 15.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[27] 16.0 sec   600 KBytes 4.92 Mbits/sec  2.382 ms 419/ 837 50%
[27] 17.0 sec   593 KBytes 4.86 Mbits/sec  3.029 ms 432/ 845 51%
[27] 18.0 sec   601 KBytes 4.93 Mbits/sec  2.238 ms 415/ 834 50%
[27] 19.0 sec   597 KBytes 4.89 Mbits/sec  3.235 ms 450/ 866 52%
[27] 20.0 sec   587 KBytes 4.81 Mbits/sec  1.521 ms 447/ 856 52%
[27] 21.0 sec   593 KBytes 4.86 Mbits/sec  3.073 ms 421/ 834 50%
[27] 22.0 sec   597 KBytes 4.89 Mbits/sec  1.854 ms 429/ 845 51%
[27] 23.0 sec   597 KBytes 4.89 Mbits/sec  3.151 ms 452/ 868 52%
[27] 24.0 sec   600 KBytes 4.92 Mbits/sec  1.945 ms 416/ 834 50%
[27] 25.0 sec   594 KBytes 4.87 Mbits/sec  3.191 ms 458/ 872 53%
[27] 26.0 sec   533 KBytes 4.36 Mbits/sec  3.098 ms 397/ 768 52%
[27] 27.0 sec   564 KBytes 4.62 Mbits/sec  3.244 ms 534/ 927 58%
[27] 28.0 sec   594 KBytes 4.87 Mbits/sec  2.754 ms 433/ 847 51%
[27] 29.0 sec   597 KBytes 4.89 Mbits/sec  2.054 ms 428/ 844 51%
[27] 30.0 sec   591 KBytes 4.85 Mbits/sec  2.131 ms 428/ 840 51%
[27] 31.0 sec   593 KBytes 4.86 Mbits/sec  1.981 ms 436/ 849 51%
[27] 32.0 sec   587 KBytes 4.81 Mbits/sec  2.080 ms 440/ 849 52%
[27] 33.0 sec   596 KBytes 4.88 Mbits/sec  3.022 ms 459/ 874 53%
[27] 34.0 sec   593 KBytes 4.86 Mbits/sec  3.029 ms 432/ 845 51%
[27] 35.0 sec   597 KBytes 4.89 Mbits/sec  2.568 ms 426/ 842 51%
[27] 36.0 sec   589 KBytes 4.82 Mbits/sec  2.158 ms 439/ 849 52%
[27] 37.0 sec   596 KBytes 4.88 Mbits/sec  1.840 ms 431/ 846 51%
[27] 38.0 sec   484 KBytes 3.96 Mbits/sec  2.158 ms 359/ 696 52%
[27] 39.0 sec   708 KBytes 5.80 Mbits/sec  2.300 ms 432/ 845 51%
[27] 40.0 sec   601 KBytes 4.93 Mbits/sec  3.281 ms 443/ 862 51%
[27] 41.0 sec   600 KBytes 4.92 Mbits/sec  2.382 ms 419/ 837 50%
[27] 42.0 sec   599 KBytes 4.90 Mbits/sec  2.104 ms 426/ 843 51%
[27] 43.0 sec   604 KBytes 4.95 Mbits/sec  2.722 ms 444/ 865 51%
[27] 44.0 sec   596 KBytes 4.88 Mbits/sec  2.017 ms 422/ 837 50%
[27] 45.0 sec   594 KBytes 4.87 Mbits/sec  3.619 ms 456/ 870 52%
[28] 46.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 47.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 48.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 49.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 50.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 51.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 52.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 53.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 54.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 55.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 56.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 57.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 58.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 59.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
[28] 60.0 sec  0.00 KBytes 0.00 Mbits/sec  0.000 ms  0/  0  0 %
```

Traffic Exchange between h11&h22 with priority enabled

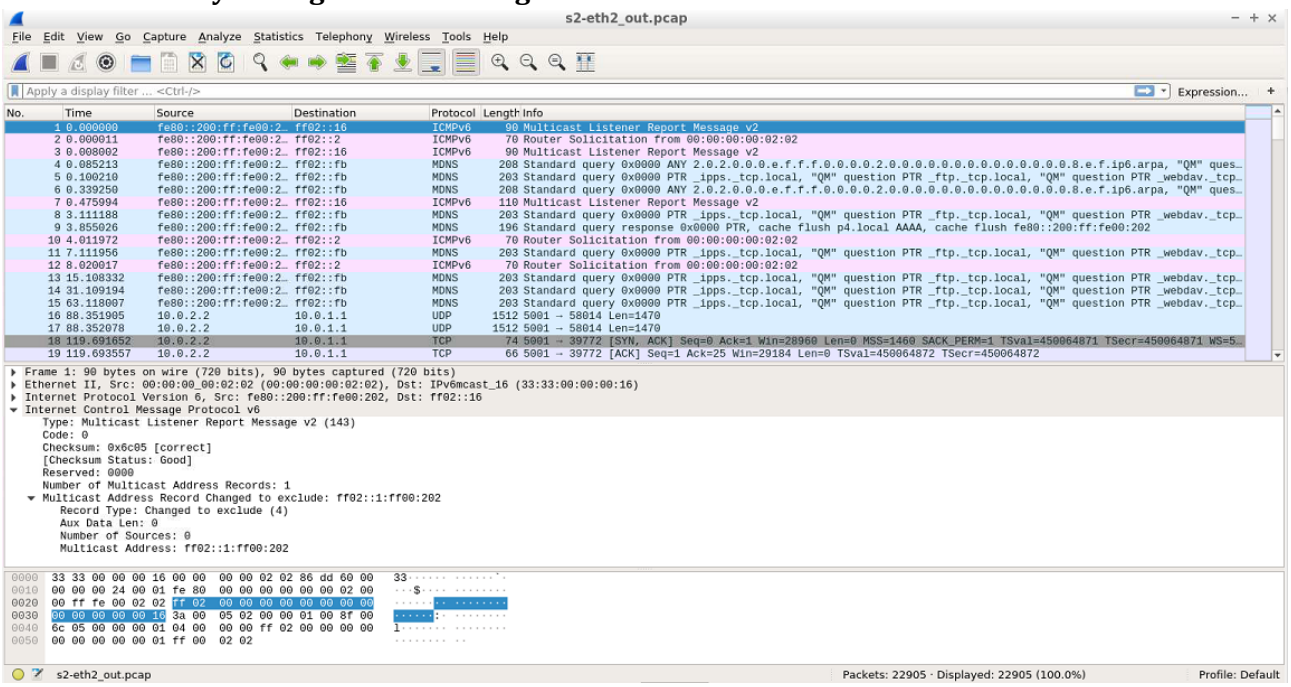
```
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 27] local 10.0.2.22 port 5001 connected with 10.0.1.11 port 58869
[ ID] Interval  Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[27] 01.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 02.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 03.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 04.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 05.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 06.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 07.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 08.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 09.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 10.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 11.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 12.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 13.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 14.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 15.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[27] 16.0 sec  1.19 MBytes 10.0 Mbits/sec  0.051 ms  0/ 850 0 %
[27] 17.0 sec  1.13 MBytes 9.47 Mbits/sec  0.263 ms  0/ 805 2.4 %
[27] 18.0 sec  1.19 MBytes 10.0 Mbits/sec  0.030 ms  0/ 851 0 %
[27] 19.0 sec  1.19 MBytes 9.98 Mbits/sec  0.341 ms  1/ 850 0.12 %
[27] 20.0 sec  1.19 MBytes 9.98 Mbits/sec  0.091 ms  1/ 850 0.12 %
[27] 21.0 sec  1.19 MBytes 10.0 Mbits/sec  0.250 ms  0/ 851 0 %
[27] 22.0 sec  1.19 MBytes 9.98 Mbits/sec  0.165 ms  1/ 850 0.12 %
[27] 23.0 sec  1.19 MBytes 9.98 Mbits/sec  0.330 ms  1/ 850 0.12 %
[27] 24.0 sec  1.19 MBytes 10.0 Mbits/sec  0.075 ms  0/ 851 0 %
[27] 25.0 sec  1.19 MBytes 9.98 Mbits/sec  0.170 ms  1/ 850 0.12 %
[27] 26.0 sec  1.19 MBytes 10.0 Mbits/sec  0.277 ms  0/ 850 0 %
[27] 27.0 sec  1.16 MBytes 9.76 Mbits/sec  0.186 ms 20/ 850 2.4 %
[27] 28.0 sec  1.19 MBytes 10.0 Mbits/sec  0.355 ms  0/ 851 0 %
[27] 29.0 sec  1.19 MBytes 10.0 Mbits/sec  0.051 ms  0/ 850 0 %
[27] 30.0 sec  1.19 MBytes 10.0 Mbits/sec  0.084 ms  0/ 851 0 %
[27] 31.0 sec  1.19 MBytes 10.0 Mbits/sec  0.226 ms  0/ 850 0 %
[27] 32.0 sec  1.19 MBytes 10.0 Mbits/sec  0.245 ms  0/ 850 0 %
[27] 33.0 sec  1.19 MBytes 10.0 Mbits/sec  0.309 ms  0/ 850 0 %
[27] 34.0 sec  1.19 MBytes 10.0 Mbits/sec  0.302 ms  0/ 851 0 %
[27] 35.0 sec  1.19 MBytes 10.0 Mbits/sec  0.480 ms  0/ 850 0 %
[27] 36.0 sec  1.19 MBytes 10.0 Mbits/sec  0.307 ms  0/ 851 0 %
[27] 37.0 sec  1.19 MBytes 9.98 Mbits/sec  0.071 ms  1/ 850 0.12 %
[27] 38.0 sec  1.19 MBytes 10.0 Mbits/sec  0.056 ms  0/ 850 0 %
[27] 39.0 sec  1.19 MBytes 10.0 Mbits/sec  0.043 ms  0/ 851 0 %
[27] 40.0 sec  1.19 MBytes 10.0 Mbits/sec  0.068 ms  0/ 850 0 %
[27] 41.0 sec  1.19 MBytes 10.0 Mbits/sec  0.045 ms  0/ 850 0 %
[27] 42.0 sec  1.19 MBytes 10.0 Mbits/sec  0.039 ms  0/ 851 0 %
[27] 43.0 sec  1.19 MBytes 10.0 Mbits/sec  0.344 ms  0/ 850 0 %
[27] 44.0 sec  1.19 MBytes 9.98 Mbits/sec  0.275 ms  1/ 850 0.12 %
[27] 45.0 sec  1.19 MBytes 9.98 Mbits/sec  0.141 ms  2/ 851 0.24 %
[27] 46.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 47.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 48.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 49.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 50.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 51.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 52.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 53.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 54.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 55.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 56.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 57.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 58.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 59.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
[28] 60.0 sec  0.00 MBytes 0.00 Mbits/sec  0.000 ms  0/  0 %
```

C.2 TSN Network Implementation Wireshark Latency Measurements Data

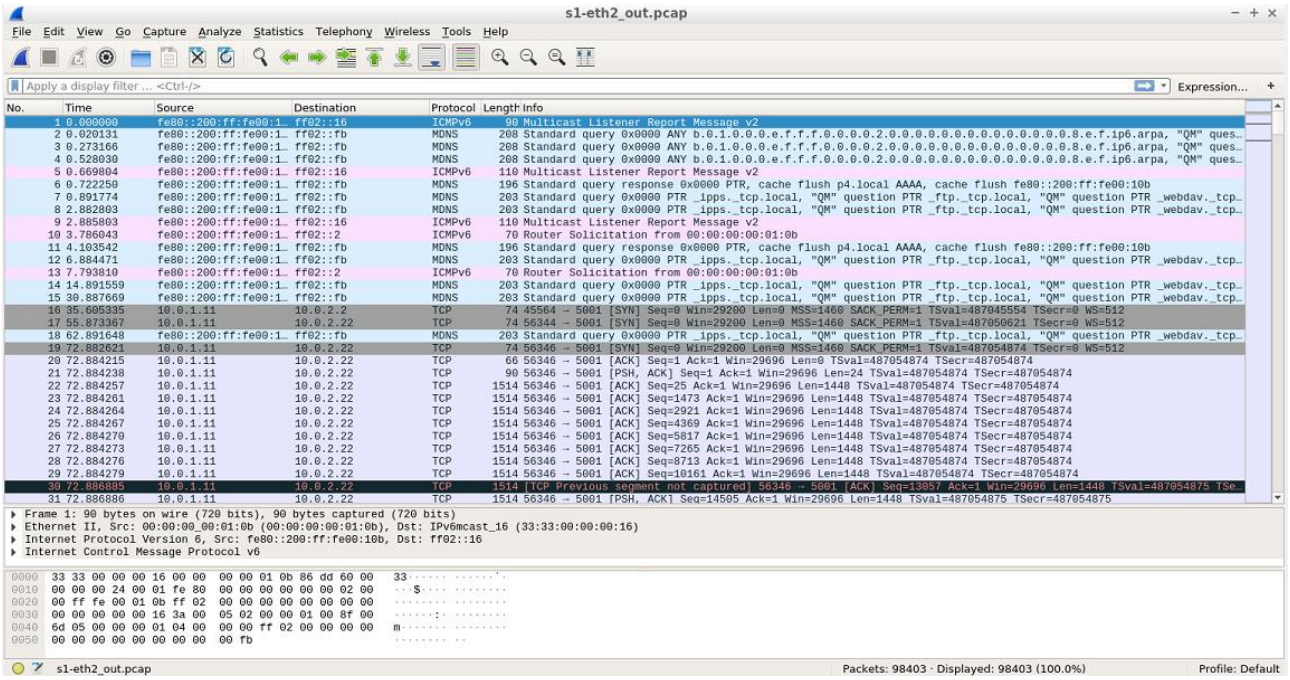
Switch 1 Latency during traffic exchange between h1 and h2



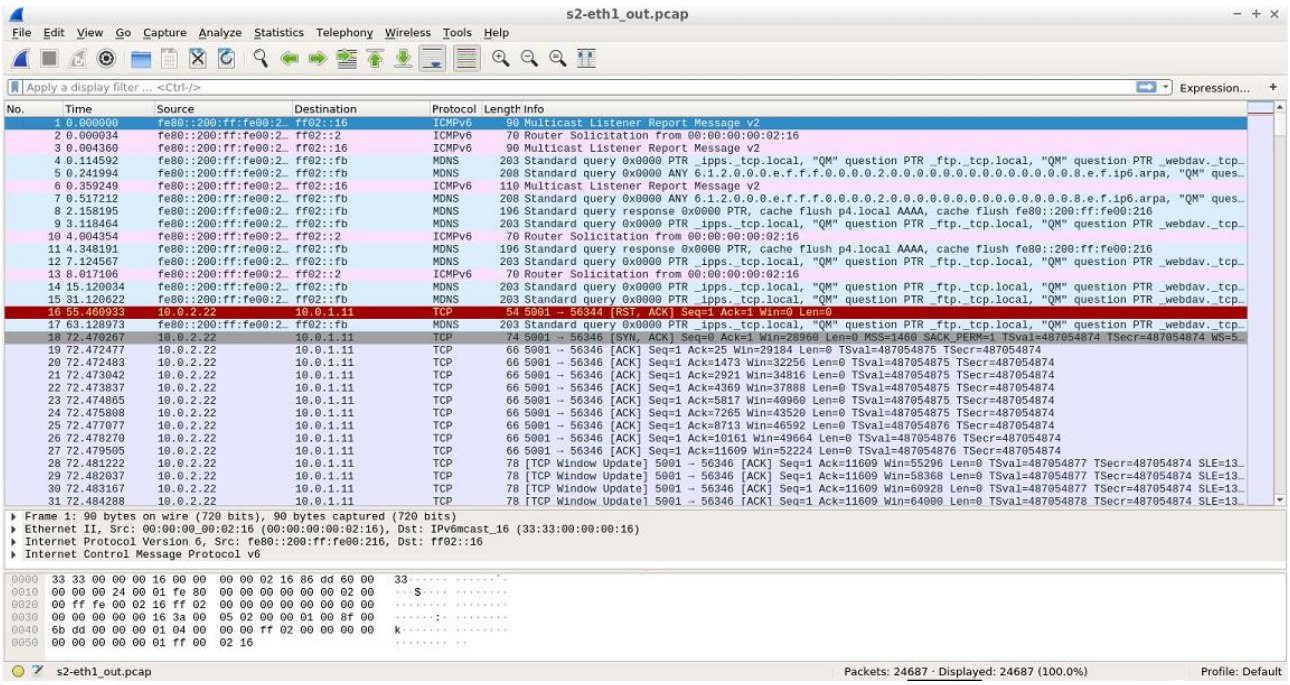
Switch 2 Latency during traffic exchange between h1 and h2



Switch 1 Latency during traffic exchange between h11 and h22



Switch 2 Latency during traffic exchange between h11 and h22



10. EBE Faculty: Assessment of Ethics in Research Projects

Project Title

A Time Sensitive Network Switch Implementation using P4

08/17/2020

id. 17251422

by **John Mkwebo** in **EBE Electrical Submissions Undergraduate**

mkwjoh004@myuct.ac.za

Original submission

08/17/2020

Project Aims	**The aim of this project is to design and implement a programmable virtual switch based on the specifications of a TSN switch and then implement a TSN network comprised of various TSN switches and a network controller.**
Ethical Issues	**There is no ethical issues that need explaining or associated with this project.**
Application Checklist	**Read the EBE Ethics in Research Handbook before completing this application** **Questionnaire to be used in the research (where applicable)** **Consent form where (where applicable see Addendum 2)** **If needed a letter motivating an expedited review. This letter should be included in the cover letter.**
Researcher(s)	**John Mkwebo**
Department	**Electrical Engineering**
E-mail	**mkwjoh004@myuct.ac.za**
Status of Applicant	**Student**
Degree Being Studied (For Students Only)	**BscEng in Electrical Engineering**
Name of Supervisor (For Students Only)	**Joyce Mwangama**
Review Track	**Normal**
Motivation for an Expedited Review	n/a

Completed Ethics Application Form

EBE_EiRC_signature_form.pdf

SECTION 1: Overview of ethics issues in your research project	n/a
Question 1: Harm to Third Parties	**No**
Question 2: Human Subjects as Sources of Data	**No**
Question 3: Participation or Provision of Services To Communities	**No**
Question 4: Conflicts of Interest	**No**
	If you have answered YES to any of the above questions, please ensure that you append a copy of your Research Proposal (Addendum 1), as well as any interview schedules or questionnaires and consent documentation (Addendum 2) and complete further addenda as appropriate. **I hereby undertake to carry out my research in such a way that:** **1. there is no apparent legal objection to the nature or the method of research; and** **2. the research will not compromise staff or students or the other responsibilities of the University;** **3. the stated objective will be achieved, and the findings will have a high degree of validity;** **4. limitations and alternative interpretations will be considered;** **5. the findings could be subject to peer review and publicly available; and** **6. I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism**
ADDENDUM 1 Supporting documents	n/a
Research Proposal **Research_Proposal.pdf**	

ADDENDUM 2 To be completed if you answered YES to question 2 in section 1	**It is required that you read the UCT Code for Research involving Human Subjects in order to be able to answer the questions in this addendum.** **Ethical research should safeguard the interests of society and the welfare of all who participate in the research, be they individuals or groups. In this section the researcher is asked to consider the implications of their research on participants in the research. The researcher should outline risks that participants will face by being involved in the research.** **When a research involves vulnerable people, a researcher is expected to obtain informed consent from participants. This informed consent should be signed by the participants. Informed consent is intended to protect the interest of both participants and the researcher should something go wrong or should conflict arise between the researcher and the participant.**
Question 2.1: Discrimination	n/a
Question 2.2: Participation of socially or physically vulnerable people	n/a
Question 2.3: Informed consent	n/a
Question 2.4: Confidentiality	n/a
Question 2.5: Anonymity	n/a
Question 2.6: Risks of physical, psychological or social harm	n/a
Question 2.7: Payments and giving of gifts	n/a
Interview Schedule	n/a
Consent Form	n/a
Additional Comments	n/a

ADDENDUM 3 To be completed if you answered YES to question 3 in section 1	**Research may sometimes interfere with the organization, progress or advancement of communities. In this section the researcher is asked to consider the effect of their research on a community or communities involved in the research. Attention should be paid to whether the research will disrupt or interrupt the normal activities of the community and how the research will influence communities in the long term.**
Question 3.1: Community participation	n/a
Question 3.2: Termination of economic or social support	n/a
Question 3.3: Provision of sub standard services	n/a
Additional Comments	n/a
ADDENDUM 4 To be completed if you answered YES to question 4 in section 1	**A conflict of interest may compromise the conduct or outcome of a research project. It may also infringe on the interests of other researchers. In this section the researcher is asked to consider if their research may be compromised by the inclusion of certain individuals or groups in the research. The researcher is also asked to consider whether the inclusion of certain individuals or groups in the research will compromise the research of others at the university. For example, if any participants in the proposed research project are also involved in other projects at the university, have you considered if this participation will negatively affect their work?**
Question 4.1: Conflicts of interest	n/a
Question 4.2: Sharing of information	n/a
Question 4.3: Conflict of interest with other research	n/a
Additional Comments	n/a