

Analysis of bandwidth reservation strategy for AVB SR classes A and B in an Ethernet TSN switch

Anders Gaustad



Thesis submitted for the degree of
Master's in Informatics: Programming and Networks
60 credits

Department of Informatics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2021

Analysis of bandwidth reservation strategy for AVB SR classes A and B in an Ethernet TSN switch

Anders Gaustad

© Anders Gaustad

2021

Analysis of bandwidth reservation strategy for AVB SR classes A and B in an Ethernet TSN switch

<http://www.duo.uio.no/>

Abstract

IEEE 802.1 Time-Sensitive Networking (TSN) is a promising Real-Time Ethernet (RTE) solution, mainly targeted at industrial automation, where real-time capabilities and reliability is of paramount importance. TSN can be seen as a set of standards, where one can choose depending on the needs of the application. Amongst these standards are IEEE 802.1Qav, defining a Credit-Based Shaper (CBS) algorithm, and traffic classes A and B. This thesis looks at a recent bandwidth reservation analysis for the CBS classes, in a single Ethernet TSN switch. The recent analysis assumes a fixed send interval (period) for each source belonging to class A and B, and imposes a deadline requirement. Through several simulation experiments this thesis looks at what happens when there is variations in the send interval of the sources. Results show that even if the average send interval is at or above the send interval used when calculating the bandwidth reservation values, according to the recent analysis, there might still occur a deadline breach. In order to secure the deadline, the bandwidth must be reserved based on the lower bound of the send interval variation range experienced by the stream.

Contents

Chapter 1 Introduction	1
Chapter 2 Background	3
2.1 Ethernet	4
2.2 Audio Video Bridging (AVB)	4
2.2.1 The Credit-Based Shaper	6
2.3 Time-Sensitive Networking (TSN)	17
2.4 Related Work	23
Chapter 3 Method	24
3.1 Selecting a simulator tool	25
3.2 OMNeT++	25
3.3 NeSTiNg (Network Simulator for Time-Sensitive Networking)	29
3.4 Modifications made to the simulator tool	33
3.5 Statistics Collection in OMNeT++	36
3.6 Result analysis using Python	38
Chapter 4 Single node analysis	40
4.1 Bandwidth calculations	40
4.2 Worst-case calculations	42
4.3 Worst-case simulation	50
4.4 Discussing the feasibility of accurate time synchronization	52
4.5 Simulating variations in send interval	53
4.5.1 Sim 1: $\pm 10\%$ variation in send interval	54
4.5.2 Sim 2: $\pm 5\%$ variation in send interval	56
4.5.3 Sim 3: $\pm 2\%$ variation in send interval	60
4.6 Further examining send interval variations	62
4.7 Sim 4: Increasing the bandwidth reservation	66
4.7.1 Addressing the low-priority traffic	68
4.8 Sim 5: constant low-priority traffic	70
4.9 Preemption of low-priority traffic	75
Chapter 5 Conclusion	78
Chapter 6 Future work	80
Appendix A Underlying work	87
Appendix B Experiment 1 calculations	92
Appendix C Event logs for scheduling scenarios	99

Appendix D	Additional simulation results.....	112
Appendix E	Queue time Calculations	118

List of Figures

Figure 2.1: Typical Ethernet frame fields, including VLAN Tag fields. Obtained from [24].	6
Figure 2.2: Output port of an Ethernet switch implementing priority scheduling, AVB/CBS and TSN. Obtained from [27].	7
Figure 2.3: 802.1Qav Credit-Based Shaper example process. Obtained from [27].	9
Figure 2.4: Credit-Based Queuing Algorithm. Obtained from [29].	11
Figure 2.5: Frame arrivals at the input port of switch X	13
Figure 2.6: Frame departures at the output port of switch X.	13
Figure 2.7: Credit-based shaped operation. Obtained from [19].	15
Figure 2.8: Timed transmission gates and preemption. Obtained from [12].	20
Figure 2.9: Packet replication and elimination. Obtained from [26].	21
Figure 3.1: Simple and compound modules. Obtained from [56].	26
Figure 3.2: Process of building and running simulation programs. Obtained from [56].	27
Figure 3.3: The architecture of OMNeT++ simulations. Obtained from [56].	28
Figure 3.4: Components of a NeSTiNg TSN switch with 4 connections.	30
Figure 3.5: Inside the Ethernet interface module.	30
Figure 3.6: Inside the queue component.	31
Figure 3.7: Configuration of transmission selection algorithm. Screenshot from .ini file.	32
Figure 3.8: Fraction of the link speed assigned to queues 5 and 6, when using CBS.	33
Figure 3.9: Original transmissionTime function located in CreditBasedShaper.cc, starting on line 103.	33
Figure 3.10: Modifications made in Ethernet.h.	34
Figure 3.11: Random function used to generate pseudorandom start times and send intervals for the sources.	34
Figure 3.12: Random function used to generate pseudorandom start times in EtherTrafGen.cc	34
Figure 3.13: Random function used to generate pseudorandom start times in EtherTrafGen.cc	35
Figure 3.14: Rule-based result analysis.	37
Figure 3.15: Using OMNeT++'s scavetool to convert a .vec result file to a .csv file.	38
Figure 3.16: Example of how I have used Jupyter Notebook to analyze result files from OMNeT++.	39
Figure 4.1: Generic construction of worst-case for stream M. Obtained from [62].	43
Figure 4.2: Worst-case scheduling scenario for stream H at a frame size of 142 B.	45
Figure 4.3: Illustrating the effect of interfering traffic on a credit-based shaper traffic class.	46
Figure 4.4: Worst-case scheduling scenario for stream M at a frame size of 142 B.	47
Figure 4.5: Worst-case scheduling scenario for stream M and H, at a frame size of 142 B.	48
Figure 4.6: Topology of Experiment 1 from [8], as implemented into OMNeT++/INET/ NeSTiNg...	50
Figure 4.7: queue time variations for stream M at a frame size of 142 B, from Sim 1, using default seed-set. Axes are in seconds.	62
Figure 4.8: queue time variations for stream M at a frame size of 142 B, from Sim 1, using default seed-set. Axes are in seconds.	62
Figure 4.9: Calculation example, based on Sim 1.	64
Figure 4.10: Sim 5 topology.	70
Figure 4.11: Number of frames in queue 4 during Sim 5.	72

List of Tables

Table 2.1: Main standards in IEEE Audio-Video Bridging	5
Table 2.2: Description of abbreviations used in the credit-based queuing algorithm. Obtained from [29].	12
Table 2.3: Time-sensitive networking standards and projects. Obtained from [12].....	17
Table 4.1: Calculated bandwidth reservation values for stream H, according to Algorithm 1 in [8]....	41
Table 4.2: Calculated bandwidth reservation values for stream M, according to Algorithm 2 in [8]. ..	42
Table 4.3: Calculated worst-case delays for stream H and M.	49
Table 4.4: Worst-case delay and average delay for streams H and M, as recorded when running worst-case scheduling scenarios.	51
Table 4.5: Start times generated for each source using the default seed-set and range.....	54
Table 4.6: Start times generated for each source using the seed-set 1.	54
Table 4.7: Send interval average for each source, using default seed-set.	54
Table 4.8: Maximum delay and average delay times for Sim 1. Using a send interval variation of $1000 \pm 10\%$, with default seed-set.	55
Table 4.9: Send interval average for each source, using default seed-set.	56
Table 4.10: Maximum delay and average delay times for Sim 2. Using a send interval variation of $1000 \pm 5\%$, with default seed-set.	57
Table 4.11: Send interval average for each source, using seed-set 1.	58
Table 4.12: Maximum delay and average delay times for Sim 2. Using a send interval variation of $1000 \pm 5\%$, with seed-set 1.....	58
Table 4.13: Send interval average for each source, using default seed-set.	60
Table 4.14: Maximum delay and average delay times for Sim 3. Using a send interval variation of $1000 \pm 2\%$, with default seed-set.	60
Table 4.15: Increased bandwidth reservation values, together with maximum delay and average delay times when using the increased bandwidth reservation values.	67
Table 4.16: Sim 1 (default seed-set) stream L statistics.	68
Table 4.17: Sim 1 (default seed-set) stream L data. Using increased bandwidth reservation values for stream H and M, shown in Table 4.15.....	69
Table 4.18: Send-intervals generated for the first two cycles from Sim 1.	71
Table 4.19: Send-intervals generated for the first two cycles from Sim 5.	71
Table 4.20: Send interval averages for Sim 5, default seed-set, range [900,1100].	72
Table 4.21: Maximum delay and average delay times for Sim 5, using default seed-set.....	73
Table 4.22: Maximum delay and average delay times for Sim 1 modified.....	74
Table 4.23: Data for Sim 5 with preemption: streams H and M may preempt L frames.	76
Table 5.1: Sim 1 with preemption: streams H and M may preempt L traffic.....	118

Preface

This thesis concludes my Master's degree at the Department of Informatics at the University of Oslo. This thesis was done as part of the collaboration between University of Oslo and Western Norway University of Applied Sciences. I would like to thank my supervisors, Professor Øivind Kure at the University of Oslo, and Professor Knut Øvsthus at the Western Norway University of Applied Sciences.

Chapter 1

Introduction

First introduced in the 1980s, Industrial networks have since experienced major growth and development. These networks play a key role when it comes to factory and process automation systems where requirements such as reliability, determinism, and real time are especially important, in contrast to commercial networks. According to [1], Industrial networks represent one of the most important innovations of the last decades in the context of factory and process automation systems. The requirements within industrial communication have changed over the past decades, this, in combination with emerging communication and information technologies has led to different classes of industrial networks being developed; mainly the original fieldbus systems, real-time Ethernet (RTE), and wireless networks [1].

Recently there has been a growing number of devices connected to each other by using Internet protocols, while several definitions exist, this is often referred to as Internet of Things (IoT) [2]. This technology has helped facilitate a smart and connected world through consumer applications, e.g., smart homes and smart transportation. Building on the advancements in information and communication technologies such as IoT and industrial networks there has emerged a new term, «Industry 4.0» [3]. The vision of industry 4.0 is to automate and reduce the need for human intervention wherever possible by integrating smart computing and network technologies in industrial production and manufacturing. Because of this, the requirements in terms of response time, network latency, and reliability are extremely important [4]. The extension of IoT into the industrial environment as Industrial Internet of Things (IIoT) takes aim at the interconnection of anything, anywhere, and at any time in the manufacturing context [4].

With industry 4.0 and IIoT on the rise, the role played by industrial networks is of increasing importance, it is expected that these networks will be used in new ways yet to come, and that their market share will increase. Several analyses within the context of industrial networks have been carried out, the data presented in [1] shows that fieldbus systems, recently having the majority of the market share, now have been overtaken by RTE networks and are expected to gradually decline. RTE networks are expected to grow considerably the coming years,

while industrial wireless networks, currently having the significantly lowest market share, are expected to have the highest annual growth [1]. RTE networks are described in [5] as the most important communication platforms in automation technology, and further confirms that RTE networks are replacing conventional fieldbus systems, as previously stated in [1], because of high performance and availability of Ethernet technology. [6] also states network scalability in regard to number of devices as a reason why RTE has overtaken fieldbus systems. Network scalability becomes especially important in the context of IIoT, envisioning networks consisting of hundreds or even thousands of devices[6].

When IT became widely used in everyday life around the millennium, this stimulated emerging Ethernet-based networks in automation that borrowed technology from the IT world. Since Ethernet lacked real-time capabilities, this led to the development of dedicated solutions by the industry [3]; changes on several OSI layers were done in order to solve the lack of real-time in a variety of ways [7]. Currently there are a multitude of mutually incompatible RTE implementations [3]. According to [7]: “A system is called RT capable if the response of a system to a request does not exceed a given time limit”. Time-sensitive networking (TSN) is a promising RTE solution currently under development by the IEEE 802.1 TSN task group, consisting of a set of standards, standard amendments, and projects. These standards can be seen as a toolbox, where one can pick and choose in order to fit the needs of the applications. The potential for industrial automation applications is appealing since it offers reduced latencies and accurate determinism, independence from physical transmission rates, fault tolerance without additional hardware, support for higher security and safety, and interoperability of solutions from different manufacturers [3]. TSN might be the answer to the scattered RTE scene [1].

Chapter 2

Background

The work performed in this thesis is based on the work performed in [8], which looks at a single Time-Sensitive Networking (TSN) switch, more specifically it looks at the Credit-Based Shaper (CBS) functionality at the output port of the switch. In short, [8] provides an algorithm for determining the minimum bandwidth reservation for the credit-based shaper traffic classes A and B in a single Ethernet TSN switch, where each traffic class is given a deadline constraint. This minimum bandwidth reservation allows each frames in streams A and B to meet the deadline, even if the worst-case scheduling scenario occurs at the output port of the switch. Note, this bandwidth reservation analysis only applies at the output port of a TSN switch, forwarding class A, B, and best-effort traffic.

During a research review it became apparent that [8] provided possibility for further research, as it looks at a single output port of a TSN switch, providing opportunity to extend their work to a larger network. In addition, [8] *only* provides an algorithm for calculating bandwidth reservations for AVB classes A and B. In this thesis I look at how the calculated bandwidth reservations affect average delay times, but also at what happens when I change the assumption made in [8] about the sources being completely synchronized to the rest of the network (fixed period), as this might not always be realistic. Also, [8] does not discuss the amount of L traffic in their analysis, it only assumes that an L frame is present in the worst-case. This thesis also looks at how the reserved bandwidth for streams H and M affect L traffic, and how changing the L traffic affects the delay times for streams H and M.

Note that in order to understand the work performed in this thesis, it is required to have a good understanding of the work performed in [8], I therefore provide a detailed summary of their work in Appendix A.

The CBS is defined as part of the Audio Video Bridging (AVB) standard which builds upon the functionality of Ethernet. And TSN builds upon the functionality of AVB. Based on this, it makes sense to provide some theory for Ethernet, AVB and TSN, with the main theory concentrated on AVB and the CBS as this is where most of the work is performed.

2.1 Ethernet

Ethernet is ubiquitous, especially in the office and home networking environments, but Ethernet has also established itself as a widely accepted technology in the field of industrial automation [9]. This can largely be attributed to the availability of widely accepted standards, adherence to these standards allow for interoperability between devices of different vendors, giving new customers flexibility in the choice of equipment and thereby reducing cost [10]. In addition to its broad commercialization, Ethernet offers high bandwidth, something that also makes it attractive for industrial communication [8]. However, Ethernet is not suitable for real-time and safety critical applications due to the lack of real-time capabilities [3], [11]. Mixing multiple flows would lead to greater latency and packet delay variation (jitter), this would limit the use cases for time-sensitive traffic [12].

Traffic differentiation (TD) is the act of prioritizing or degrading specific types of traffic over others, this can be done based on traffic content, protocol, origin, destination, etc. [13]. TD was introduced in 1998 in the IEEE 802.1D standard, using traffic classes for differentiating traffic through priorities. Separation of different streams of traffic on the same LAN was also introduced in 1998, through virtual LANs (VLANs) in the IEEE 802.1Q standard, providing another means to achieve TD. The introduction of these mechanisms made Ethernet better suited for the transport of traffic flows with different requirements, such as multimedia streams; audio and video could be transmitted using different priorities to distinguish their requirements. However, Ethernet was not able to provide guarantees for upper time bounds to all priorities of all data streams [3], enter Audio Video Bridging.

2.2 Audio Video Bridging (AVB)

An Apple architect, Michael Johas Teener, wanted to bring high-quality audio and video to a larger market through new technologies, he made modifications to an existing standard, IEEE 1588, wherein precision clock synchronization was defined. This resulted in 802.1AS being developed, in order to have synchronization at the heart of layer-2 switches. Based on this work, a new IEEE task group was formed to standardize 802.1AS in 2005, initially called Audio-Video Bridging (AVB) [12]. The goals of AVB, as listed in [14]:

- provide a network-wide precision clock reference;

- limit network delays to a well-known (and hopefully small) value;
- keep non-time sensitive traffic from messing things up.

To this end, AVB uses four IEEE 802.1 standards, listen in the table below:

Designation	Title
802.1BA	Audio Video Bridging systems
802.1Qat	Stream Reservation Protocol (SRP)
802.1AS	Timing and Synchronization for Time-Sensitive Applications
802.1Qav	Forwarding and Queuing Enhancements for Time-Sensitive Applications

Table 2.1: Main standards in IEEE Audio-Video Bridging

IEEE 802.1BA ([15]) sets the three other standard documents into context and defines Ethernet-AVB system and default configuration that are necessary to build networks that are capable of transporting time-sensitive audio and/or video data streams. An AVB network consists of end nodes that can act as either talker or listener or both. A talker would be an end station that is the source of a stream, and a listener would then be an end station consuming the stream, a device can act as talker and listener simultaneously. In addition to end stations acting as talkers or listeners, the network also contains AVB switches, that besides offering the normal switching functionality also offer additional functionality that allows time-sensitive data to be transmitted in the network.

IEEE 802.1Qat ([16]) describes the Stream Reservation Protocol (SRP), allowing for network resources to be reserved for specific traffic streams. SRP allows stream endpoints to register their willingness to talk or listen to specific streams, SRP propagates this information through the network. The propagations of these declaration messages allow for collection of QoS information along the paths, this information can be used by AVB in order to guarantee a deterministic latency and jitter [17]. According to [18], SRP can reserve up to 75% of the bandwidth on a port for AVB traffic, the rest can then be used for best-effort traffic, note that this is not completely accurate: the IEEE Std 802.1Q ([19]) informs that the 75% limit is the default setting and the standard acknowledges that this might not be enough for some networks where for example the amount of non-reserved traffic is very low, the standard therefore states that this default setting can be changed by management. From what I can gather it does not seem like the standard imposes a maximum allowed bandwidth reservation for the stream reservation classes.

IEEE 802.1AS ([20]) Ensures that every device within an AVB network operates on a synchronous clock and is based on IEEE1588-Precision Time Protocol (PTP). This allows time sensitive applications to meet jitter, wander, and time synchronization requirements. The synchronization approach consists of mainly two parts, the selection of the «best» clock in the network, and the distribution of this clock information by using PTP. In AVB networks the time-based synchronization information is distributed from a so-called grand master along an established tree to other devices in the network [14].

2.2.1 The Credit-Based Shaper

The CBS is described in IEEE 802.1Qav: Forwarding and Queuing Enhancements for Time-Sensitive Applications ([21]). The specification defines 8 classes of traffic, 6 best-effort classes and 2 Stream Reservation (SR) traffic classes, A and B, with real time constraints [22]. Class A is for time-critical traffic with stringent time requirements, and class B is for time-critical traffic with less stringent time requirements. Class A provides a maximum delay of 2 ms over seven hops, while class B provides a maximum delay of 50 ms over 7 hops [23]. From this it follows that more hops will result in higher delay [14]. The priority level for the traffic classes goes from 0-7, where 7 has the highest priority and is normally used for traffic class A, then comes traffic class B, and 6 best-effort (BE) classes [24]. The priority is coded using three bits in the VLAN tag of incoming frames, the effect of this tagging is to place the frames in a specific queue in the receiving switch: the output queue corresponding to the priority level of the PCP value in the frames [25].

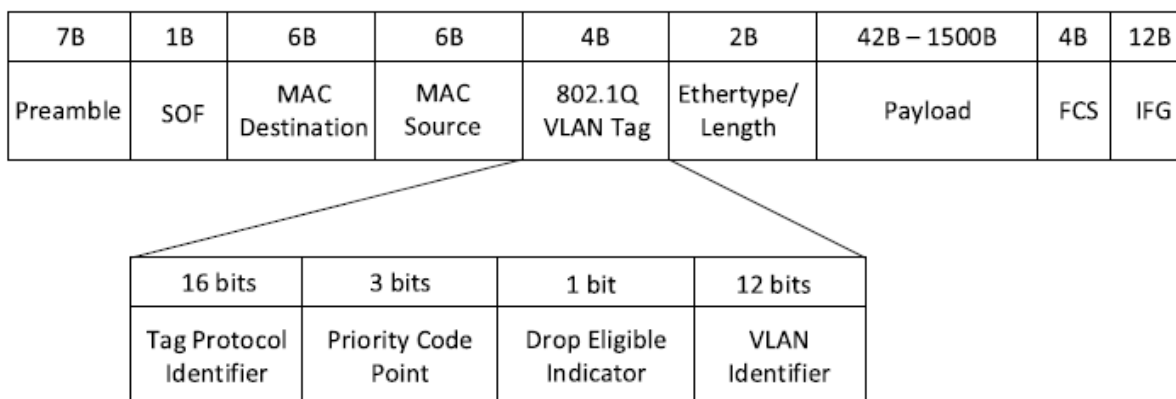


Figure 2.1: Typical Ethernet frame fields, including VLAN Tag fields. Obtained from [24].

Figure 2.1 shows the Priority Code Point (PCP) field, defined in the IEEE 802.1Q standard ([19]). These 3 bits, providing 8 possible (priority) values, correspond to the number of queues an output port of an Ethernet switch may implement [25]. In the AVB/TSN model these queues are first-in-first-out (FIFO) [26].

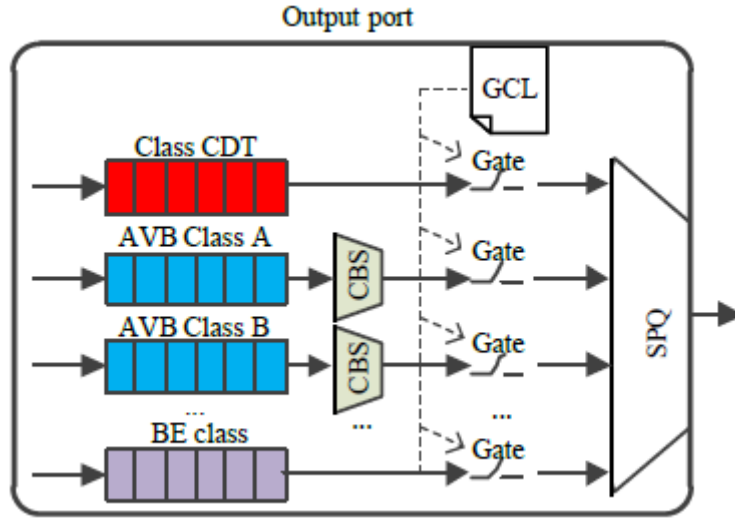


Figure 2.2: Output port of an Ethernet switch implementing priority scheduling, AVB/CBS and TSN. Obtained from [27]

The CDT traffic class and GCL/Gate seen in Figure 2.2 above is TSN functionality, not AVB, and can be disregarded for now. We can see that only SR class A and B are subject to the CBS, however both SR classes and BE classes are subject to strict priority scheduling (SPQ). In strict priority scheduling, also known as priority-based scheduling, frames in higher ranking queues have precedence over frames in lower ranking queues, this means that a non-AVB frame can only be sent if there are no frames *available* for transmission in the queues corresponding to class A and B. After a frame has entered the ingress port and been allocated to the queue corresponding to the value of the PCP field in the frame, the strict priority selection takes place before the frame can exit the switch through the egress port. But before frames reach transmission selection, a CBS algorithm for queues containing class A and B traffic is used. The CBS algorithm uses a credit value, this value dictates when a frame is allowed to be forwarded from a queue using the CBS algorithm [5]. The most concise summary of the forwarding rules for the CBS algorithm I have encountered, is given by [28], presented in points 1-5 below:

1. If the transmission line is free, the scheduler transmits a frame of the highest priority class that satisfies the conditions: a) its queue is not empty; and b) it has a non-negative credit.
2. The credit of an AVB class is reduced linearly with rate *send slope* when the class transmits.
3. The credit of an AVB class increases linearly with rate *idle slope* when the following conditions hold simultaneously for that class: a) its queue is not empty; and b) other AVB or BE classes are transmitting.
4. Whenever an AVB class has a positive credit and its queue becomes empty, the credit is set to zero; this is called a *credit reset*.
5. If the credit is negative and the queue becomes empty, the credit increases with rate *idle slope* until the zero value.

To clarify, a traffic class has a corresponding PCP value (priority), multiple sources can belong to the same traffic class, they will then have the same PCP value in the frames they send. Each PCP value (traffic class) is mapped to a queue on the output port of one or multiple switches in a network. The SR classes (AVB classes), A and B, which uses the CBS algorithm on their corresponding queues, each have an independent credit value associated with their queue. This credit value is used by the CBS algorithm as part of the decision process to determine if the queue can forward a frame or not. The factors determining the increase/decrease rate will be explained in the next section. I will now use the rules, as shown above, to explain the example presented in Figure 2.3 on the next page.

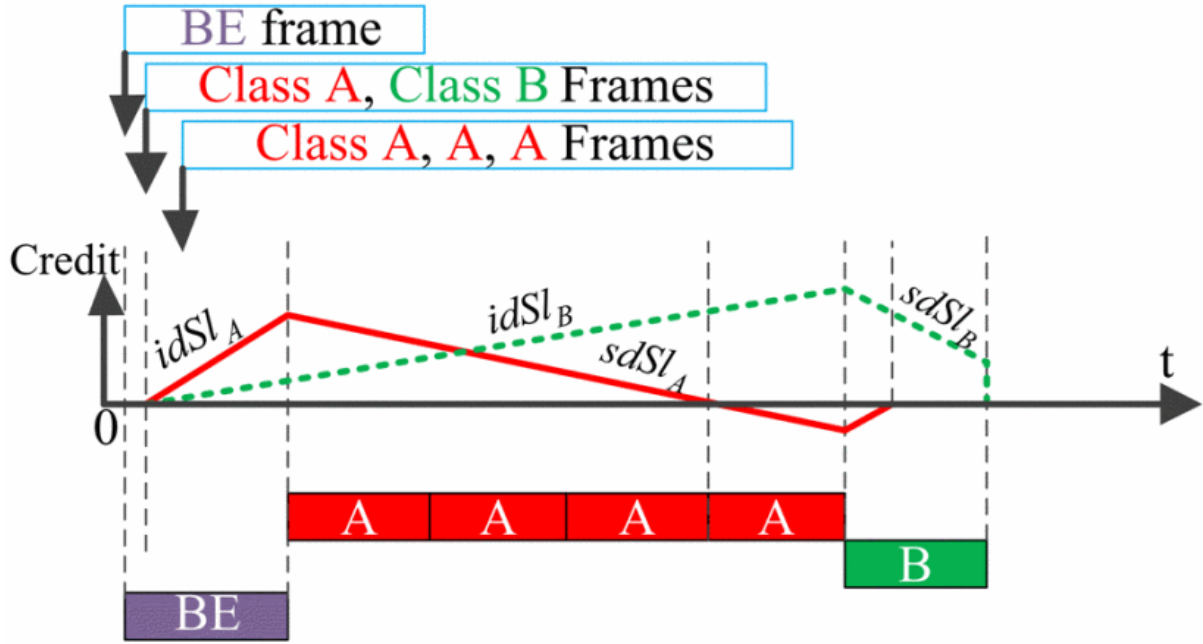


Figure 2.3: 802.1Qav Credit-Based Shaper example process. Obtained from [27].

$idSl_A$ is the idle slope for class A, and $idSl_B$ is the idle slope for class B, while $sdSl_A$ is the send slope for class A, and $sdSl_B$ is the send slope for class B. We can see that one of the BE queues starts a transmission across the outbound port of the switch right before the arrival of one class A frame and one class B frame. At this point both the class A queue and the class B queue contain a frame and the credit associated with each queue is above or equal to zero (meaning they have enough credit to transmit), however, both the class A and B queue must wait for the interfering BE frame to finish its transmission (see rule 1).

Since both the class A and B queue contain one or more frames, and since both queues have to wait for interfering traffic, the credit value can increase above 0 (see rule 3), which it does for both queues. Note that the idle slope for class A is steeper than for class B, the reason for this is because class A, in this example, has a higher bandwidth reservation than class B, which consequently means that the cost of forwarding a frame will be less for class A, reflected in the send slope for class A compared to class B. This is not explicitly stated in [27], but becomes apparent when going through the CBS information found in Annex L in IEEE 802.1Q, [19].

When the BE frame has finished transmitting across the outbound port of the switch, both the class A and B queue have a credit value above or equal to zero, but the class A queue will forward first since it has a higher priority value (see rule 1). Note that there arrived 3 more

class A frames during the transmission of the BE frame. Since the class A queue still has a positive credit after the first frame forward it can forward 3 more frames until the credit becomes negative, at this point the class A queue is empty, but had there been another frame in the queue it could not have been forwarded since the credit value of the queue is negative (see rule 1). After the final frame forward from the class A queue the credit increases back to zero at rate idle slope (see rule 5). At this point the class B queue can forward since it currently is the highest priority class that meets all the conditions in rule 1.

On the next page I show the CBS algorithm with an accompanying table containing descriptions for abbreviations used in the algorithm.

Require: $bwF, pTxRate, maxIS$

```

1:  $credit \leftarrow 0$ 
2:  $idleSlope \leftarrow (bwF \cdot pTxRate)$ 
3:  $sendSlope \leftarrow (idleSlope - pTxRate)$ 
4:  $hiCredit \leftarrow (idleSlope \cdot maxIS / pTxRate)$ 
5:  $lowCredit \leftarrow (sendSlope \cdot maxFSize / pTxRate)$ 
6: loop
7:   if  $credit \geq 0$  then
8:      $bTx \leftarrow true$ 
9:   else
10:     $bTx \leftarrow false$ 
11:   if  $ConFP \neq true$  then
12:     if  $noQFs > 0$  AND  $bTx$  then
13:        $frame = Queue.Get(i)$ 
14:        $SendFrame(frame)$ 
15:        $transmitting \leftarrow true$ 
16:   if  $frameServiced$  then
17:      $\triangleright$  this implies 't' is  $(lFrameSize / pTxRate)$ 
18:      $transmitting \leftarrow false$ 
19:   if  $transmitting \neq true$  then
20:     if  $noQFs > 0$  then
21:        $credit \leftarrow credit + idleSlope \cdot t$ 
22:       if  $credit \geq hiCredit$  then
23:          $credit \leftarrow hiCredit$ 
24:     else
25:        $credit \leftarrow credit - sendSlope \cdot t$ 
26:       if  $credit \leq lowCredit$  then
27:          $credit \leftarrow lowCredit$ 
28:   if  $frameServiced$  AND  $noQFs = 0$  AND  $credit > 0$  then
29:      $credit \leftarrow 0$ 

```

Figure 2.4: Credit-Based Queuing Algorithm. Obtained from [29].

Abbreviation	Description
pTxRate	The port transmission rate [bps].
bwF ¹⁾	The bandwidth fraction of the pTxRate that is available to the queue [bps].
mIS ¹⁾	The maximum interference size, [bits] that can delay the transmission of a time sensitive frame.
credit	The transmission credit, [bits], that is currently available to the queue.
transmitting	TRUE for the duration of a frame transmission from the queue; FALSE when any frame transmission from the queue has completed.
idleSlope	The rate of change of credit [bps]. when the value of credit is increasing. (i.e., while transmitting is FALSE).
sendSlope	The rate of change of credit, in bits per second, when the value of credit is decreasing (i.e., while transmitting is TRUE).
transmitAllowed / bTx	TRUE when the credit parameter is zero or positive; FALSE when the credit parameter is negative.
MaxFSize _z	The maximum sized frame that can be transmitted through the port for the traffic class Z concerned.
FSize _z	The frame size for the traffic class Z concerned.
hiCredit	The maximum value that can be accumulated in the credit parameter.
loCredit	The minimum value that can be accumulated in the credit parameter.
ConFP	Indicate the presence of conflicting frames.
noQFs	Number of queued frames.
lFrameSize	Size of last frame in the current burst.
frameServiced	When the port completes the transmission of the last frame in a burst.
$\sum_{j < z}$	Used to indicate the sum of the values of all SR classes with high priority (and therefore, earlier letters in the collating sequence) then Z.

Table 2.2: Description of abbreviations used in the credit-based queuing algorithm. Obtained from [29].

The CBS is a way of achieving traffic shaping; it reduces bursts by spreading the packets out in time, which in turn will reduce stress on the downstream bridges, allowing for shorter queues and limiting overall latency of AVB streams [12]. Shorter queues translates to a reduction in memory demand in the bridges [14]. According to [30], the CBS algorithm “prevents the starvation of lower priority flows”, while the reason is not explicitly explained, the reason can be seen when looking at the example in Figure 2.3. I have also experienced this when running simulation experiments: with only strict priority scheduling, if there are always frames in the higher priority queues the lower priority queues will never be allowed to transmit. But, when using the CBS algorithm, the lower priority queues can forward when the credit of the queues using the CBS becomes negative, and thereby not starve.

Below I provide a generic illustration in order to show the traffic shaping effect of the CBS (spreading the frames out in time), also making it easier to see why it prevents the starvation of lower priority streams:

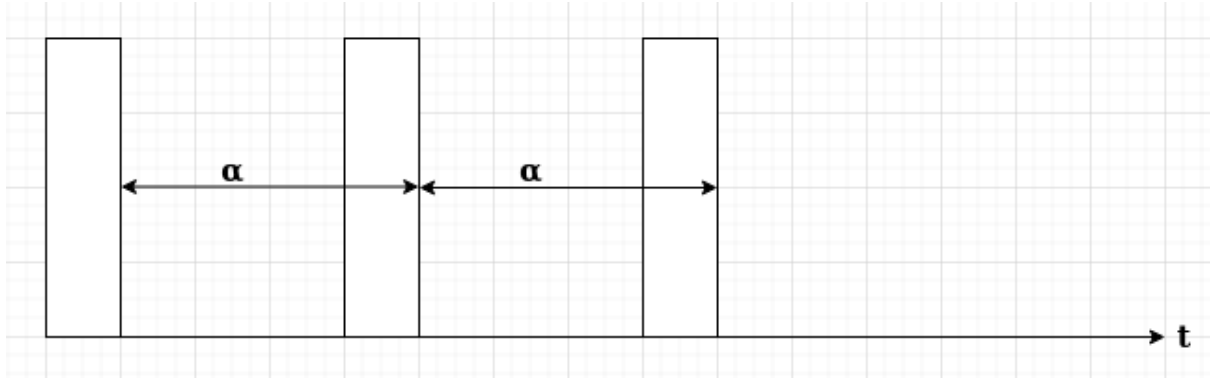


Figure 2.5: Frame arrivals at the input port of switch X

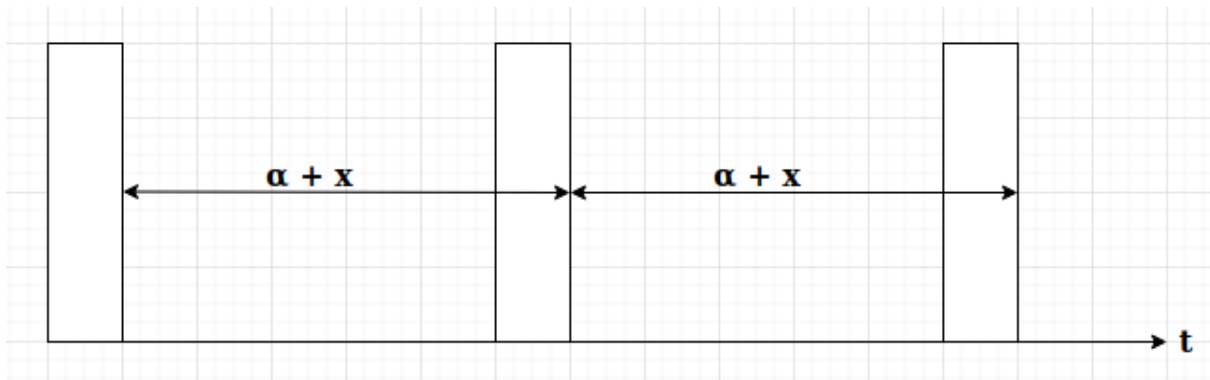


Figure 2.6: Frame departures at the output port of switch X

During this added time, x , as a result of negative credit value, lower priority frames can transmit.

2.2.1.1 Credit-Based Shaper Calculations

The information provided in this section is from the 802.1Q standard, [19], mainly from Annex L. I will focus on the calculations for `hiCredit`, `loCredit`, `idleSlope` and `sendSlope`, both because I use these calculation methods later in my thesis, but also because it gives a better understanding of how the CBS works.

According to [19]: “The credit-based shaper algorithm has a single externally determined parameter, `idleSlope`, that determines the maximum fraction of the `portTransmitRate` that is

available to the queue associated with a traffic class (*bandwidthFraction*), as follows in Equation (L-1)". The L-1 equation is shown below:

$$\textit{bandwidthFraction} = \textit{idleSlope} / \textit{portTransmitRate}$$

As an example, this means that if I have a *portTransmitRate* of 100 Mb/s and I have assigned SR class A to queue 7 and SR class B to queue 6, then I can assign queue 7 (class A) a bandwidth of 50 Mb/s and class B (queue 6) a bandwidth of 25 Mb/s by assigning queue 7 a *bandwidthFraction* of 0.5 and queue 6 a *bandwidthFraction* 0.25. This also means that the *idleSlope* for class A (queue 7) is now 50 Mb/s, and the *idleSlope* for class B is 25 Mb/s. Note that "if a given traffic class does not have a frame available for transmission, then a lower priority traffic class that does have a frame available for transmission is able to transmit. Hence, if a given traffic class that uses the credit-based shaper algorithm has been allocated X% of the available bandwidth on a Port, but only uses (X-Y)%, then the unused portion of its allocation (Y%) is available for other traffic classes to use if they are in a position to do so." [19].

From the definitions in [19]:

idleSlope is "the rate of change of credit, in bits per second, when the value of credit is increasing."

sendSlope is "the rate of change of credit, in bits per second, when the value of credit is decreasing." And is given by:

$$\textit{sendSlope} = (\textit{idleSlope} - \textit{portTransmitRate})$$

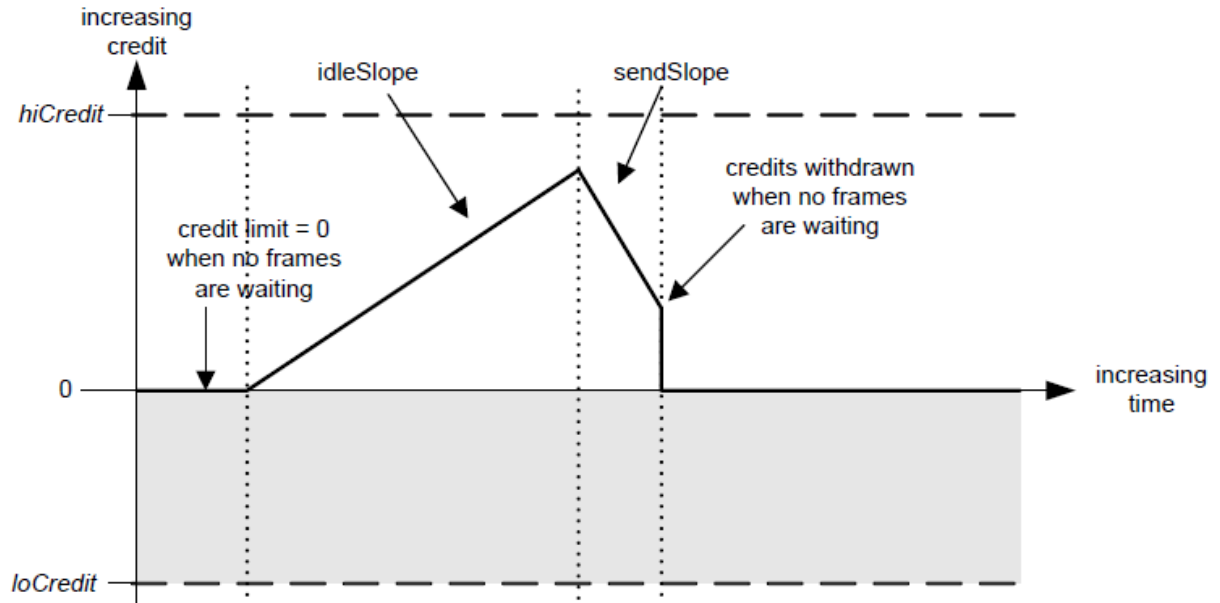


Figure 2.7: Credit-based shaped operation. Obtained from [19].

There is also an upper limit for the credit value, *hiCredit*, and a lower value, *loCredit*, as shown in Figure 2.5. [19] provides two equations for calculating these values, as shown below:

$$loCredit = maxFrameSize \times (sendSlope/portTransmitRate) \quad (L-2)$$

$$hiCredit = maxInterferenceSize \times (idleSlope/portTransmitRate) \quad (L-3)$$

Calculating the *hiCredit* value is straightforward as the maximum interference size (*maxInterferenceSize*) is simply the size of one maximum sized Ethernet frame, which is 2000 octets, meaning $2000 \times 8 \text{ bits} = 16000 \text{ bits}$. In the case of a 100 Mb/s Ethernet connection and a traffic class (queue) with 50% bandwidth reservation, *hiCredit* will then be 50% of 2000 octets: $16000 \text{ bits} \times (50 \text{ Mb/s} / 100 \text{ Mb/s}) = 16000 \times 0.5 = 8000 \text{ bits}$. In this example I am assuming the port transmit rate is equal to the link speed.

In order to calculate *loCredit* I need to find the *maxFrameSize*, which is the maximum number of bits that can be transmitted in a frame for a stream (traffic class). This value is defined through the SRP protocol, which uses a traffic specification (TSpec) for each stream. In addition to defining the maximum number of bits per frame, the TSpec also defines a maximum number of frames which can be transmitted (*maxIntervalFrames*). Both the *maxFrameSize* and the *maxIntervalFrames* are measured over a class measurement interval which applies at the source (talker) of the stream. For class A this class measurement interval

is 125 μ s and for class B it is 250 μ s. The purpose of the class measurement interval is to define acceptable talker behavior [19]. 8

So, going back to the example above with a link speed of 100 Mb/s and a traffic class with 50% bandwidth reservation. If I am using traffic class A, then maxFrameSize will be 50% of the maximum number of bits I can transmit during 125 μ s over a 100 Mb/s link. Which is: $100\,000\,000\text{ b/s} \times 0,000125\text{ s} \times 0.5 = 6250\text{ bits}$. However, according to [19], this number needs to be an integral number of octets, as 6250 is not, the number is rounded down to 6248. Having found the maxFrameSize I now need to find the sendSlope. Knowing the idleSlope is equal to the reserved bandwidth of the class (queue), 50 Mb/s, then $\text{sendSlope} = 50\text{ Mb/s} - 100\text{ Mb/s} = -50\text{ Mb/s}$, which is the rate of change of credit per second when transmitting a frame across the output port. loCredit, given by the L-2 equation on the previous page, is therefore equal to $6248\text{ bits} \times -0.5 = -3124\text{ bits}$.

Using the example above, if the queue assigned to class A forwards a frame of size 842 B across the output port of the switch, then the credit cost of this will be: time it takes to transmit the frame across the output port of the switch \times sendSlope. The time it takes to transmit a 842 B frame over a 100 Mb/s link is $842 \times 8\text{ bit} / 100\,000\,000\text{ b/s} = 0,00006736\text{ s}$. The cost will then be $0,00006736\text{ s} \times -50\,000\,000\text{ b/s} = -3368\text{ bit}$. If we assume the credit was at zero before the frame forward, then afterwards it will be -3368. It will then take $3368/\text{idleSlope}$ seconds before the credit returns to zero.

2.3 Time-Sensitive Networking (TSN)

It became apparent that AVB could be used not only for audio and video transmission but also for factory automation and process control. However, while AVB does offer some real-time capabilities it does not sufficiently cover all the requirements for factory automation and process control [10]. In the application space of audio and video transmissions, such as AVB, if frames were delayed or lost the consequences were noticeable but not harmful. There was no need for an absolute guarantee for end-to-end latency for streams [10]. This is not the case for mission critical industrial and automotive applications where streams must arrive within their relative deadline [30]. The IEEE Audio-Video Bridging (AVB) task group was renamed to Time-Sensitive Networking (TSN) in 2012 to reflect this new and enlarged scope. The focus of TSN is to improve real-time capabilities and reliability of standard Ethernet. Mainly targeted at industrial automation, it takes aim at some of the shortcomings of AVB with goals of providing: 1) further reduction of latencies and more accurate determinism; 2) independence of physical data transmission rates; 3) native support for higher security and safety; 4) fault tolerance without the need for additional hardware; and 5) interoperability of solutions from different vendors [1]. According to [1], TSN can be viewed as a toolbox for IEEE 802 networks, approximately 60 IEEE standards are relevant for TSN.

Designation	Title	Incorporation	Further information
802.1Qat	Stream reservation protocol	802.1Q-2011	http://standards.ieee.org/findstds/standard/802.1Qat-2010.html
802.1Qav	Forwarding and queuing enhancements for time-sensitive streams	802.1Q-2011	http://standards.ieee.org/findstds/standard/802.1Qav-2009.html
802.3br	Interspersing express traffic	Standalone standard	http://standards.ieee.org/findstds/standard/802.3br-2016.html
802.1Qbu	Frame preemption	802.1Q-2018	http://standards.ieee.org/findstds/standard/802.1Qbu-2016.html
802.1Qbv	Enhancements for scheduled traffic	802.1Q-2018	http://standards.ieee.org/findstds/standard/802.1Qbv-2015.html
802.1Qci	Per-stream filtering and policing	802.1Q-2018	http://standards.ieee.org/findstds/standard/802.1Qci-2017.html
802.1Qch	Cyclic queuing and forwarding	802.1Q-2018	http://standards.ieee.org/findstds/standard/802.1Qch-2017.html
802.1CB	Frame replication and elimination for reliability	Standalone standard	http://standards.ieee.org/findstds/standard/802.1CB-2017.html
802.1CM	Time-sensitive networking for fronthaul	Standalone standard	https://standards.ieee.org/findstds/standard/802.1CM-2018.html
P802.1Qcr	Asynchronous traffic shaping	802.1Q amendment project	http://www.ieee802.org/1/pages/802.1cr.html
P802.1Qcc	SRP enhancements and performance improvements	802.1Q amendment project	http://www.ieee802.org/1/pages/802.1cc.html
P802.1AS-Rev	Timing and synchronization for time-sensitive applications – revision	Standalone project	http://www.ieee802.org/1/pages/802.1AS-rev.html

Table 2.3: Time-sensitive networking standards and projects. Obtained from [12].

Table 2.3 lists the main TSN standards and projects. The previously described IEEE standards 802.1Qat, 802.1Qav and 802.1AS from Table 2.1 in the AVB section in this chapter are used in TSN. Some of the most important features of TSN networks, described in [23], are:

- Synchronize of all network devices and hosts to an accuracy between 1 μ s and 10ns.
- Form contracts for each TSN flow, between the transmitter and the network. Allowing for Bounded latency and zero congestion loss.
- Flexibility in contract management; new contracts can be made, and old ones terminated.
- Achieve high reliability by replicating packets, sending them on separate paths, and then eliminating the duplicate before both packets reach the destination. Providing a way of mitigating packet loss due to equipment failure.
- Mechanisms to improve coexistence with best effort-services. Such as transmission pre-emption which reduces interference of best effort traffic on time critical traffic.

With the features above in mind, I will briefly describe some of the IEEE standards added in TSN.

The 802.1AS-2011 standard ([20]), used in AVB, provides precise time synchronization of the network nodes and can achieve a reference time accuracy better than 1 μ s. The handover procedure, should there occur a loss of the grandmaster which acts as the network wide reference time, is not instantaneous and might cause time jumps at some nodes. This led to the development of 802.1AS-Rev-Timing and synchronization for time-sensitive applications-revision ([31]). Offering improved redundancy compared to the previous version, allowing for seamless low-latency handover and recovery of time synchronization in failure nodes. Which is achieved through multiple grand master clocks and synchronization spanning trees [25].

802.1Qbv- Enhancements for scheduled traffic ([32]), defines transmission gates for the queues, and time schedules for controlling them. Since TSN targets mainly industrial automation it needs to take into consideration traffic characteristics of such networks [1]. One of those characteristics is the regular transmission of typically low bandwidth traffic streams, which is sent from controllers to actuators and from sensors to controllers, known as “control traffic”. This traffic is used for control loops, failure to send this information, or if the information is lost or delayed might cause the feedback loop to misbehave or fail [12]. In an industrial setting, loss or delay of control traffic can result in severe consequences [10]. Therefore, TSN implements schedule-driven communication, often referred to as time-triggered (TT) communication [25], and introduces a new traffic class, CDT [27]. The CDT

class is intended for the regular transmissions of control traffic and has the highest priority amongst TSN traffic classes [27]. See the previously shown Figure 2.2 to view the new CDT class in context of the previously described AVB classes.

The schedule-driven communication is realized using the synchronized local clocks in the networks, a gate corresponding to each queue, and a Gate Control List (GCL) with an open/close schedule for each gate. A schedule (GCL) is generated by the system designer and distributed to the end stations and switches as part of their configuration. This schedule tells the end stations and switches at which points in time to set the gates to the open/closed state [25]. This time schedule can be configured centrally or through a distributed mechanism. Algorithms for calculating these schedules are not part of TSN [33]. As mentioned previously, 802.1Qbv schedules the activation and deactivation of queues by introducing a gate per queue, however, 802.1Qbv only schedules the activation and deactivation of queues, not frame transmissions. When the gate is in the open state the transmission selection process may select frames from that queue [25].

Having the means to limit queues to being open only during a specific time window means that one can create a “channel” reserved for one or multiple traffic classes [12]. A downside to scheduled transmissions is that the transmission of a frame can begin within the scheduled time window but might not finish transmitting before the time window closes, this might cause frames from the queue/queues in the closing time window to delay frames from the queue/queues in the next time window [25]. A guard band at the end of each time window was introduced to tackle this problem, this guard band had the length of the longest possible frame, 1530 byte ([18]). Larger frames are often preferred because they reduce the need for overhead compared to sending several smaller frames, giving better utilization of the medium in terms of payload versus overhead. More sophisticated implementations allow for a reduction of the guard band by using a preemption mechanism [12]. Below I provide a figure showing preemption, the guard band is displayed in orange.

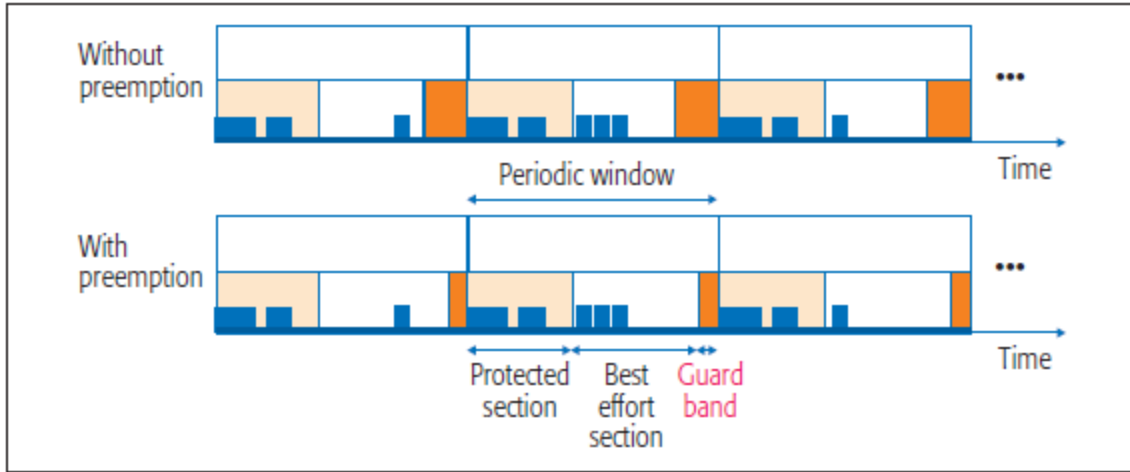


Figure 2.8: Timed transmission gates and preemption. Obtained from[12].

Transmission preemption is defined by the standards 802.1Qbu-Frame preemption ([34]) and 802.3br-Interspersing express traffic ([35]). Allowing for selection of queues on an output port as “preemptable” or “preempting”. This means that queues designated as preempting can interrupt transmissions of queues designated as preemptable. In the case a preemptable frame was interrupted, it will be resumed after the preempting queue or queues have finished transmitting [26]. Remember that the time windows are scheduled based on control traffic requirements. Control traffic cannot be delayed, it requires a stringent QoS. A guard band is therefore needed, but preemption allows for a reduction of this guard band, meaning better utilization of the bandwidth for non-control traffic [18]. When preemption is used, the guard band can be reduced to 128 bytes [27].

802.1Qbu defines procedures for holding or suspending the transmission of a frame in bridges or end stations, allowing one or several more urgent frames to transmit in front of it. The standard also defines procedures to resume sending the suspended frames, after the more urgent frames have been transmitted. While the 802.3br standard allows the splitting of a non-urgent frame into smaller fragments, so that a more urgent frame can interrupt it before its transmission is finished [25].

The 802.1CB standard-Frame replication and elimination for reliability ([36]), replicates frames and sends them on separate paths in the network. These redundant transmissions reduce frame loss. When replicating the frames, a sequence number is added, and two or more identical frame flows are created. At or near the destination, the duplicate frames are detected and eliminated. Nodes in between the sender and receiver can be configured to discard and re-

replicate frames at various points for the purpose of protection against multiple failures [26]. An example of frame replication and elimination is provided below.

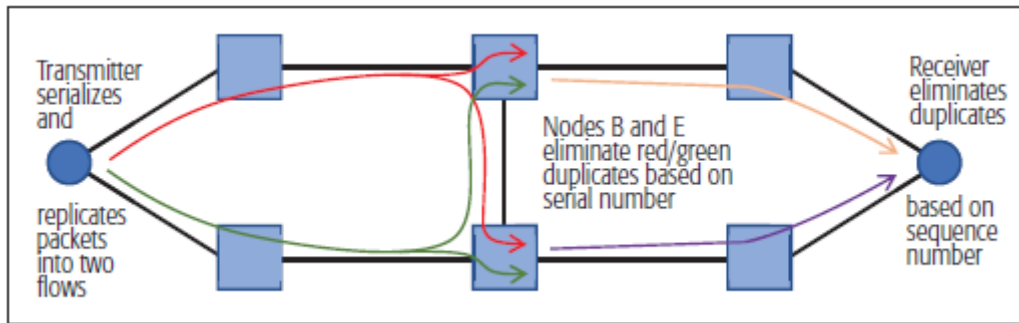


Figure 2.9: Packet replication and elimination. Obtained from [26].

“CB does not cover the creation of the multiple paths over which the duplicates are transmitted (this is realized using other protocols, such as the IEEE Std 802.1Q, IEEE Std 802.1Qca, and IEEE P802.1Qcc).” [25].

802.1Qci ([37]) provides per stream filtering and policing by using the stream-identification capabilities of 802.1CB. A traffic class for a stream is determined by an internal priority value (IPV) together with combinations of header fields such as MAC source address, destination address, VLAN and IP header fields. 802.1Qci also allows for further differentiation of streams based on when the packet was received in a time cycle and its priority value [12]. In addition, 802.1Qci provides monitoring of streams for quality-of-service protection and mitigating actions should a violation occur [25].

802.1Qch-Cyclic queuing and forwarding ([38]), describes how to use the previously mentioned standards to construct a network with bounded latency and guaranteed bandwidth for time-sensitive streams while also supporting best-effort traffic [12].

In AVB, the 802.1BA- Audio Video Bridging systems put the other AVB standard documents into context, and defined system and default configuration that are necessary to build networks capable of transporting time-sensitive audio and/or video data streams. There is no such document that explains how to control the different functionality of TSN to achieve a desired operation for different use cases [12].

The added functionality in TSN changes the forwarding rules for a queue using the CBS algorithm, the updated rules, as compared to the rules presented in section 2.2.1, are presented below:

1. If the transmission line is free, the scheduler transmits a frame of the highest priority class that satisfies all the conditions: a) its queue is not empty; b) its gate is open; and c) it has a non-negative credit if it is an AVB class.
2. The credit of the AVB class is reduced linearly with rate *send slope* if class x transmits.
3. The credit of the AVB class increases linearly with rate *idle slope* when the following conditions hold simultaneously for class x: a) its gate is open; b) its queue is not empty; and c) other AVB BE classes are transmitting.
4. The credit of an AVB class remains constant if the corresponding gate is closed and, during any additional overhead in the case of preemption mode for CDT.
5. Whenever class x has a positive credit and its queue becomes empty, the credit is set to zero; this is called *credit reset*.
6. If the credit is negative and the queue becomes empty, the credit increases with rate *idle slope* until the zero value.

The rules presented above were obtained from [28].

2.4 Related Work

The main focus of this thesis is how maximum delay and average delay times, for queues using the credit-based shaper algorithm, are affected by variations in frame arrival rates to said queues, in a TSN switch. Specifically, queues using bandwidth reservations as calculated by the bandwidth reservation algorithm presented in [8]. At the time of writing this, [8] is cited by 6 articles ([39],[40],[41],[42],[43],[44]), neither of which present work similar to the work in this thesis. I have not been able to find articles/papers presenting work similar to the work performed in this thesis.

For a detailed summary of the work performed in [8], which is required in order to understand the work performed in this thesis, I refer to Appendix A. Also, as reference number 8 will be mentioned several times throughout this thesis, oftentimes together with other references, I have chosen to mark it in cursive text for the remainder of this thesis, making it easier to keep track of which reference is the underlying work.

Chapter 3

Method

In order to understand the chosen method(s) of evaluation, I first present what this thesis aimed to look at. This thesis wanted to look at the bandwidth reservation algorithm in [8], and at what happens when the assumption made about the sources having a fixed period is changed, with a focus on average delay times and maximum delay. This thesis also wanted to look at how the reserved bandwidth for streams H and M affect L traffic, and how changing the L traffic pattern affects the delay times for streams H and M. Another ambition was expanding the single node analysis from [8] into a larger network, while this ambition was not reached, some extrapolations from single node analysis to multi node analysis can be made, and is discussed in a later chapter.

In my analysis I use one of the theoretical experiments presented in [8], Experiment 1, and I confirm that each source in streams H and M is able to meet the deadline requirement, given the worst-case scheduling scenario, for this topology, occurring at the output port of the switch. Note that [8] does not explicitly show the worst-case scenario given the topology of the experiments, nor does it list the exact bandwidth reservation values calculated for each experiment. Therefore, I have calculated the bandwidth reservation values for Experiment 1, according to their presented algorithm, and I have used underlying research from [8] to find the worst-case scheduling scenario for streams H and M.

Three main methods can be used for evaluation: experiments with real network components, formal network analysis, and network simulations. The equipment needed for real network experiments is costly and I do not have access to the required equipment. Confirming that the calculated bandwidth reservation values are sufficient for streams H and M to meet the deadline requirement in the worst-case scheduling scenario, given the topology of Experiment 1, can be done using formal network analysis: network calculus, there is no need for a simulation tool. However, when looking at average delay times for the different streams, and when introducing variability in the period of the sources, using network calculus becomes more difficult. In addition, when expanding the analysis from [8] to a larger network, taking into account that I want the possibility of analyzing TSN functionality, such as frame preemption, I find network simulations much more practical. Because of this, I have chosen to

perform the worst-case analysis by using calculations, and the remainder of the analysis, as mentioned above, is done using a simulator tool.

3.1 Selecting a simulator tool

There are two well-known simulator tools for simulating network behavior, ns-3 [45] and OMNeT++ [46]. When it comes to simulating TSN functionality, the prevalent choice in academia clearly seems to be OMNeT++ ([47], [48], [49], [18], [50], [33], [51], [52], [53]). However, many of the presented simulation models simulating TSN functionality is either not made publicly available or is made to simulate only specific TSN functionality.

My choice of simulation model is NeSTiNg (Network Simulator for Time-Sensitive Networking) [54], together with the OMNeT++/INET discrete event simulation framework. Based on my research, this seems to be the most complete simulation model/framework available for my purpose, providing both CBS functionality, as well as TSN functionality for the possibility of extending the work done in [8] further. NeSTiNg is presented in [33], and additional use of NeSTiNg is shown in [55].

3.2 OMNeT++

OMNeT++ is a C++ based, object-oriented, discrete event network simulator and framework. As listed in the OMNeT++ Simulation Manual, [56], its use cases include:

- modeling of wired and wireless communication networks
- protocol modeling
- modeling of queueing networks
- modeling of multiprocessors and other distributed hardware systems
- validating of hardware architectures
- evaluating performance aspects of complex software systems
- in general, modeling and simulation of any system where the discrete event approach is suitable, and can be conveniently mapped into entities communicating by exchanging messages.

According to [57]: “The motivation of developing OMNeT++ was to produce a powerful open-source discrete event simulation tool that can be used by academic, educational and research-oriented commercial institutions for the simulation of computer networks and distributed or parallel systems.”

When modelling a system as discrete events, state changes (events) occur at discrete instances in time. Events does not take any time to happen, meaning multiple events can happen at the exact same time. With this type of system modelling, one assumes that nothing interesting occurs between two events, such as a state change where a frame goes from transmitting to a different state. Example of events can be start of a frame transmission and end of a frame transmission [56]. “the definition of “interesting” events and states always depends on the intent and purpose of the modeler” [56].

It is important to note that OMNeT++ by itself *only* provides a framework and the tools required for writing simulations, it does not directly provide the simulation components. Also, “specific application areas are supported by various simulation models and frameworks such as the Mobility Framework or the INET framework” [57]. Where INET is an independently developed model library, providing models for the Internet stack such as TCP, UDP, IPv4, wired and wireless link layer protocols, different link components, etc. [58]. A multitude of different simulation models have been developed, for areas such as wireless and ad-hoc networks, sensor networks, IP and IPv6 networks, MPLS, wireless channels, peer-to-peer networks, optical networks, and more [57].

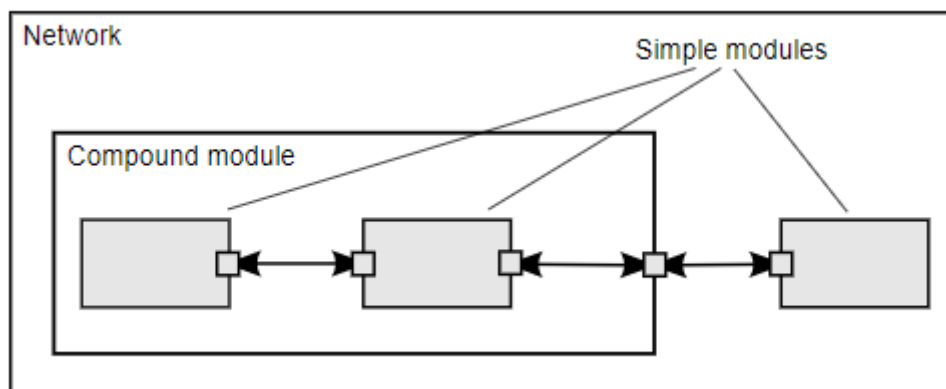


Figure 3.1: Simple and compound modules. Obtained from [56].

The previously mentioned simulation models consist of modules, and there are two different types of modules, seen in Figure 3.1, both are instances of module types, and the number of hierarchy levels in terms of grouping simple modules into compound modules and so forth is

unlimited [56]. These modules communicate with each other using message passing, and the messages, can be sent either to the next connected module, or directly to the end module. An input gate and an output gate is used as input and output interface, respectively, for modules, and the messages sent between modules can contain arbitrary data [57]. When these messages arrive at their destination module, this marks the occurrence of an event, and the time when the message arrives at the destination module (arrival time) is the time when the event occurs. The future events are stored in a data structure, and processing one event may cause the insertion of a new event into the data structure or the deletion of an already stored event [56].

OMNeT++ provides a topology description language, NED, this is used to define the structure of the model (the modules and their interconnections). Using this language, the user can also designate some compound modules as networks, which are self-contained simulation models, they have no gates to the external world. OMNeT++ also provides a graphical editor which uses NED as its native file format, where one can create and edit the network structure [57].

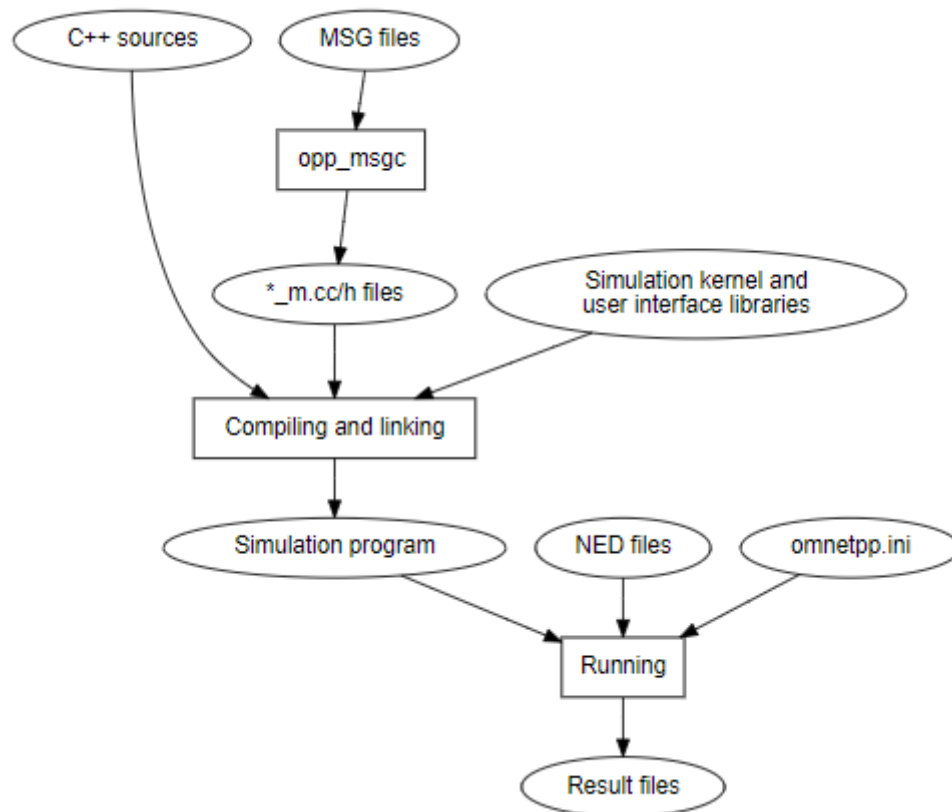


Figure 3.2: Process of building and running simulation programs. Obtained from [56].

Simple modules have the .cc/.h suffix and are written in C++. Message files, having the .msg suffix, contains message definitions which are translated into C++ classes. NED files (.ned), contain component declaration and topology descriptions. And .ini files are configuration files containing model parameters and other settings [56]. The below points, directly quoted from [56], lists the process of turning source into an executable form.

1. Message files are translated into C++ using the message compiler, *opp_msgc*
2. C++ sources are compiled into object form (.o files)
3. Object files are linked with the simulation kernel and other libraries to get an executable or a shared library

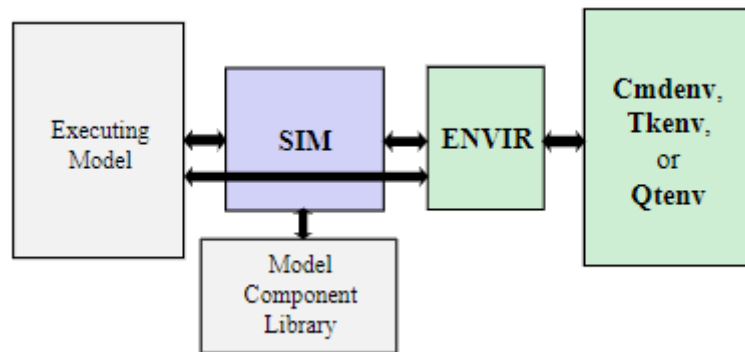


Figure 3.3: The architecture of OMNeT++ simulations. Obtained from [56].

Executing Model contains objects of different types such as modules, channels, etc., which are all instances of the components in the Model Component Library. The Executing Model talks directly with the SIM (simulation kernel and class library), which manages future events and activates modules in the Executing Model as the events occur, in turn the SIM also responds to function calls from the Executing Model which uses classes in the library of the SIM. The Executing Model also talks to ENVIR, which is a library, and contains the *main()* function (where the execution of the simulation program begins). ENVIR decides what happens in the simulation program, and instructs SIM in which models to set up for the simulation, while also supplying SIM with module parameters. Cmdenv, Tkenv and Qtenv are user interfaces, and which one to use when running a simulation program is determined by the *main()* function in ENVIR. ENVIR is also used by SIM to write output statistics such as vector files and to print debug information [56].

Cmdenv is a command line user interface, primarily for batch execution. It can be used to pass arguments or configuration options to the configuration (.ini) file. Tkenv and Qtenv are graphical user interfaces, allowing one to get a more detailed picture of the simulation at any point in time, in order to better view what is happening inside the network. “You would typically test and debug your simulation under Tkenv or Qtenv, then run actual simulation experiments from the command line or shell script, using Cmdenv.” [56]

3.3 NeSTiNg (Network Simulator for Time-Sensitive Networking)

NeSTiNg is a simulation model for Time-Sensitive Networking (TSN), It uses the OMNeT++ framework, together with the INET framework and enhances it with TSN-capable components. NeSTiNg is open source and is available here [54] together with an install guide. It is currently tested with OMNeT++ version 5.5.1 and INET version 4.1.2 under Linux. For my simulation experiments I have used the tested OMNeT++/INET versions with Ubuntu 18.04.5 LTS Operating System.

Below I will show screenshots of a TSN switch, as implemented in NeSTiNg, when running a simulation in the Qtenv graphical runtime environment. I will then describe how a frame is processed by the TSN switch.

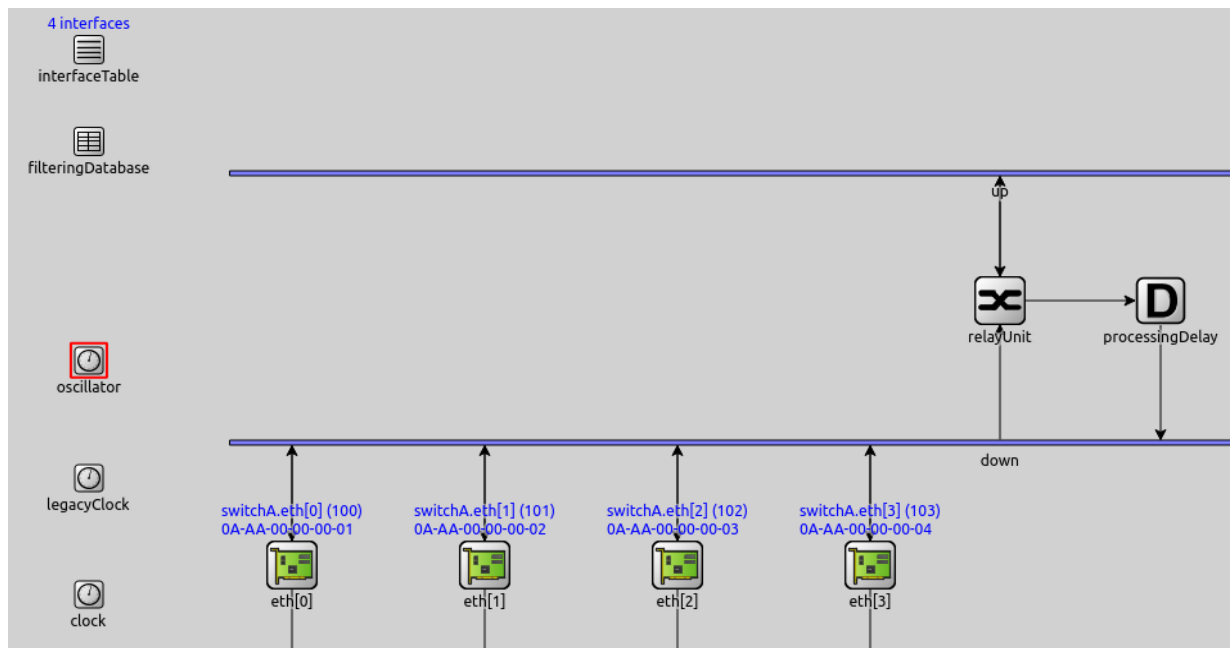


Figure 3.4: Components of a NeSTiNg TSN switch with 4 connections.

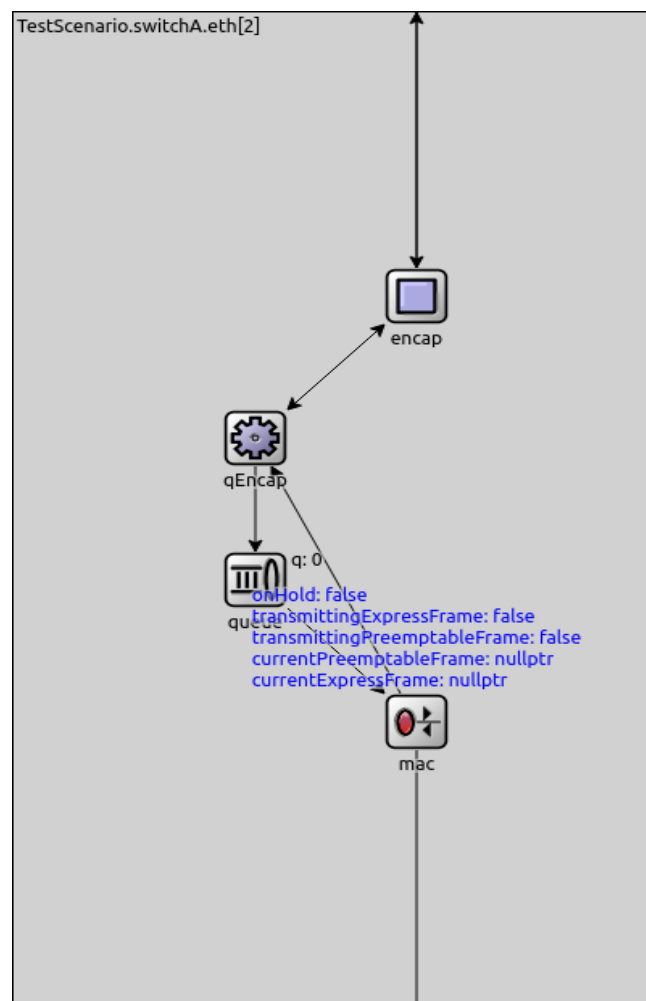


Figure 3.5: Inside the Ethernet interface module.

As stated earlier, NeSTiNg uses the modules already available in INET, such as the *mac* component which is connected to the link component. In addition to handing the frames over to the link component, the *mac* component models the transmission delay [33].

When a frame arrives at an input port of the switch it is processed by the *mac* component, and sent up to the *relayUnit* component which sends the frame to the *processingDelay* component (see Figure 3.4). The length of the processing delay can be configured in the .ini file and is meant to simulate the processing delay in a switch. After this the frame is sent to the output Ethernet interface, which contains the *queue* component (Figure 3.5) of the egress port.

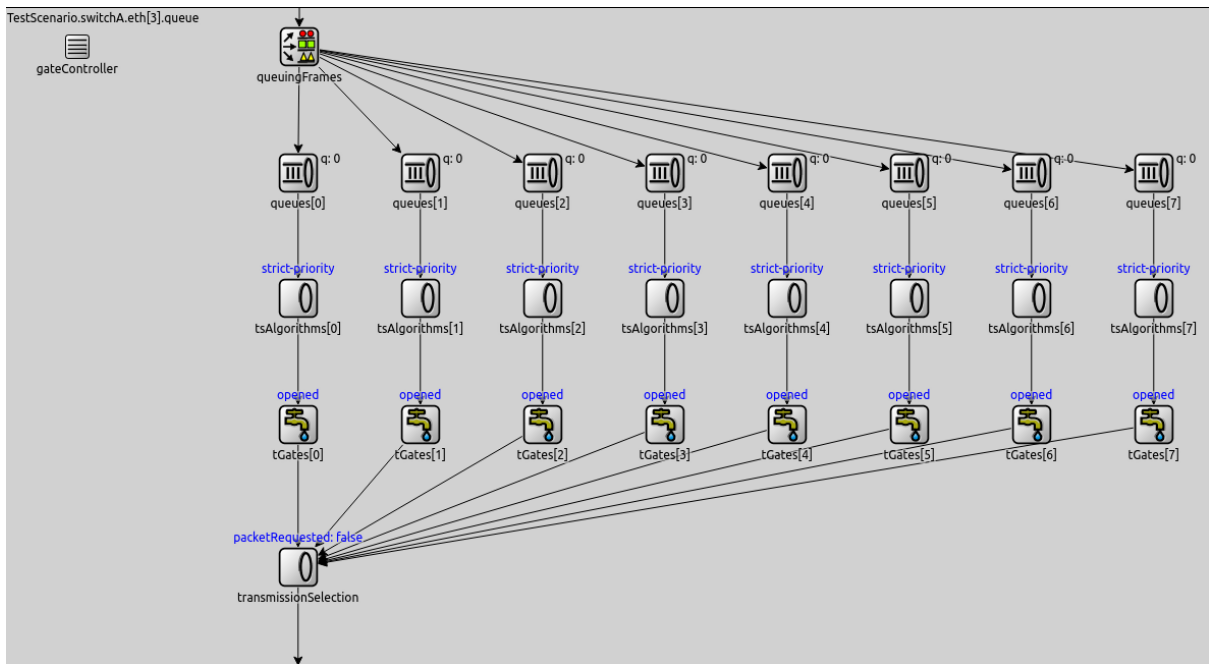


Figure 3.6: Inside the queue component.

First the frame arrives at the *queuingFrames* component, here the PCP field of the frame is evaluated, and based on the PCP value the frame is placed in one of the eight queues. Which frame is eligible for transmission is decided by the *transmissionSelection* component. There are two triggers for when the *transmissionSelection* component queries a queue for a frame [33]:

1. Downward control flow path, where information about frames ready for transmission is propagated downwards. This triggers the processing of frames newly arrived in the queue.
2. Upward control flow path, after the lower layer completes a frame transmission. Note that there is an idle period after a frame has been forwarded across the output port.

NeSTiNg ensures that even though two events can be scheduled for the exact same time, the events are executed in the correct order. An example: the arrival of a high priority frame and a low priority frame at an idle output port of a switch at the exact same time. What can happen is that the transmission of the low priority frame can be scheduled before the transmission of the high priority frame, by putting the low priority arrival event in the global event schedule before the high priority arrival event. This is done by scheduling a self-message in the `transmissionSelection` component, for further reading about this see section IV in [33].

Between the queues holding the frames and the `transmissionSelection` component is the shapers. “From a software architectural perspective, the shapers provide a queue interface for the `transmissionSelection` component” [33]. As can be seen in Figure 3.6, each queue has a gate component and a transmission selection algorithm component associated with it. The gate component is part of the time-aware shaper (TAS), and the credit-based shaper can be selected via the transmission selection algorithm component. In Figure 3.6 the strict priority transmissions selection algorithm is used. The shapers decide if a frame associated with their queue is eligible for transmission or not, based on their current state.

Configuration and input data for the simulation is normally entered in the `.ini` file. This is where I select which transmission selection algorithm to use, below I show how I can configure queues 5 and 6 to use the credit-based shaper:

```

**.switch*.eth[*].queue.tsAlgorithms[0].typename = "StrictPriority"
**.switch*.eth[*].queue.tsAlgorithms[1].typename = "StrictPriority"
**.switch*.eth[*].queue.tsAlgorithms[2].typename = "StrictPriority"
**.switch*.eth[*].queue.tsAlgorithms[3].typename = "StrictPriority"
**.switch*.eth[*].queue.tsAlgorithms[4].typename = "StrictPriority"
**.switch*.eth[*].queue.tsAlgorithms[5].typename = "CreditBasedShaper"
**.switch*.eth[*].queue.tsAlgorithms[6].typename = "CreditBasedShaper"
**.switch*.eth[*].queue.tsAlgorithms[7].typename = "StrictPriority"

```

Figure 3.7: Configuration of transmission selection algorithm. Screenshot from `.ini` file.

The PCP value used by each source in its frame transmissions is configured in the `.ini` file. Using the configuration in Figure 3.7 as an example: frames with a PCP value of 6 will be placed in queue 6, and thereby be subject to credit-based shaping. Also, the fraction of the link bandwidth available to the queue, when using credit-based shaping, is configured as shown below:

```
network1_random.switch*.eth[*].queue.tsAlgorithms[6].idleSlopeFactor = 0.04544
network1_random.switch*.eth[*].queue.tsAlgorithms[5].idleSlopeFactor = 0.04544
```

Figure 3.8: Fraction of the link speed assigned to queues 5 and 6, when using CBS.

The open/close schedule for the gates is controlled by the gate control list (GCL), this is an .xml document passed as a parameter to the gate controller component of the output port of a switch. Meaning, for each output port, there is one GCL, which controls the open/close schedule for the 8 queues of the output port.

After the *transmissionSelection* component has polled the shapers, and the shapers have decided that a frame is eligible for transmission, the eligible frame is then forwarded down to the *mac* component and over the link. In addition to CBS, TAS and frame tagging (using PCP), NeSTiNg also supports frame-preemption.

3.4 Modifications made to the simulator tool

Modification 1

In the process of familiarizing myself with NeSTiNg, I discovered a discrepancy between the bandwidth assigned to a queue, when using the credit-based shaper, and the actual throughput measured for that queue. This discrepancy was the result of the credit-based shaper algorithm implemented in NeSTiNg using the wrong frame size in its spending credit calculation; the algorithm used a larger frame size than the actual frame size going over the link, when calculating the credit spent when forwarding a frame. This resulted in a lower credit value after each forward, or, if you will, a steeper send slope, meaning the time before the credit recovered to zero was longer, translating to a lower throughput than what was assigned to the queue. In order to fix this, I changed the hardcoded values in the function shown below:

```
simtime_t CreditBasedShaper::transmissionTime(Packet* packet) {
    // Ieee8021q::getFinalEthernet2FrameBitLength(packet) is somehow wrong (1704B instead of correctly 1521B + 8B PHY + IFG)
    int lengthInBits = packet->getBitLength() + (21 + 8 + 12) * 8;
    simtime_t transmissionTime = timeForCredits(getPortTransmitRate(), lengthInBits);
    return transmissionTime;
}
```

Figure 3.9: Original transmissionTime function located in CreditBasedShaper.cc, starting on line 103.

so when a queue using the credit-based shaper algorithm forwards a frame, it now uses the actual frame size going over the link when calculating the amount of credit spent forwarding that frame, in agreement with IEEE Std 802.1Q ([19]).

Modification 2

I also made a minor modification to INET: in Experiment 1 from [8] the total size of the L frames are 1542 B, by default INET allows for a maximum frame size of 1526 B. In order to implement Experiment 1 into my chosen simulator, I therefore had to change this in the Ethernet.h file:

```
const B MAX_ETHERNET_FRAME_BYTES = B(1542);
```

Figure 3.10: Modifications made in Ethernet.h

Modification 3

When introducing variability in the start times and send intervals of the sources, in Chapter 4, I use a random function:

```
/**
 * Returns a random integer with uniform distribution in the range [a,b],
 * inclusive. (Note that the function can also return b.)
 *
 * @param a, b the interval, a<b
 * @param rng index of the component RNG to use, see getRNG(int)
 */
virtual int intuniform(int a, int b, int rng=0) const {return omnetpp::intuniform(getRNG(rng), a, b);};
```

Figure 3.11: Random function used to generate pseudorandom start times and send intervals for the sources.

This random function is used in the EtherTrafGen.cc file, which is part of the *EtherTrafGen* module in INET, this module is used by each source, and is responsible for generating frames. The random function is used in the function *void EtherTrafGen::initialize(int stage)*, as such:

```
//startTime = par("startTime");
startTime = intuniform(0,1000) / 1000000.0;
```

Figure 3.12: Random function used to generate pseudorandom start times in EtherTrafGen.cc

With the parameters in Figure 3.12, the function will generate a pseudorandom whole number value in the range of 0-1000 μ s. The commented out top line is the original code, which takes the value of the *startTime* variable, as assigned in the .ini file of the simulation.

I also use the same random function in `void EtherTrafGen::scheduleNextPacket(simtime_t previous)`, in `EtherTrafGen.cc`, as such:

```
//next = previous + *sendInterval;  
gap = intuniform(900,1100) /1000000.0;  
next = previous + gap;
```

Figure 3.13: Random function used to generate pseudorandom start times in `EtherTrafGen.cc`

The variable *gap* is defined as type *double* further up in the function. The original code is commented out. With the parameters used in Figure 3.13, the function will return a pseudorandom whole number in the range 900-1100 μ s.

By using these two random functions, instead of each source having a predetermined start time and a fixed send interval, the sources now:

1. Generate a pseudorandom integer value in the interval 0-1000 μ s, used as the time for the first transmission (start time).
2. Transmit first frame at $t =$ generated value from step 1.
3. Generate a pseudorandom integer value, used as the time to schedule the next transmission.
4. Transmit a frame at the time of the previously scheduled transmission.
5. Repeat steps 3 and 4 until the simulation time limit is reached.

Note that modification 1 and 2 are in effect for all simulation experiments conducted in this thesis. Modification 3 is only used when simulating random start times and send interval times. Apart from this I use the default modules provided INET and NeSTiNg. I do, however, implement some changes in order to record relevant statistics, discussed in the section below.

Also, a feature in OMNeT++ is that the same pseudorandom numbers will be generated every time you run the simulation [56]. Meaning, when I run the simulation with a new frame size and bandwidth reservation, the functions will produce the same pseudorandom numbers in the same order as in the previous frame size and bandwidth reservation. Unless I change the seed-set.

3.5 Statistics Collection in OMNeT++

OMNeT++ allows for collection of statistics by using the *signal* mechanism. There is also the option of recording statistics directly from C++ code, using the simulation library, but this method is not preferred, as stated in [56]. The *signal* mechanism can be used in one or several modules in the simulation model, where it can expose a chosen variable for result collection. The exposed variable may then be recorded using the *@statistic* property in a chosen module. There is already implemented some statistics collection in INET and NeSTiNg, by using signals and declared statistics, such as the *queueingTime* statistic in the *LengthAwareQueue* module in NeSTiNg. This is the module shown as *queues[x]* in Figure 3.6. In addition to the already available statistics, I have also added some, such as a statistic to record the send interval values generated by the random function in the EtherTrafGen module (see Figure 3.13). Below I will show how the send interval statistic was implemented in the EtherTrafGen module, for further reading about result recording using signals and declared statistics see the OMNeT++ manual ([56]) sections 4.14, 4.15, and chapter 12.

I declared the signal in EtherTrafGen.h:

```
simsignal_t frameGapSignal;
```

and the signal is registered in the initialize function in EtherTrafGenSched.cc:

```
frameGapSignal = registerSignal("frameGap");
```

The signal, together with the variable I want to record, is emitted in the *scheduleNextPacket* function in EtherTrafGenSched.cc:

```
emit(frameGapSignal, gap);
```

In EtherTrafGenSched.ned I specify how I want to record the exposed variable:

```
@signal[frameGap](type=simtime_t; unit=s);  
@statistic[frameGap](title="frame gap"; source=frameGap; record=vector, min, max; interpolationmode=sample-hold);
```

frameGap is the name of the statistic, the title property key is not necessary but provides the option to specify a more descriptive name for the statistic, this name can also be used as legend when creating charts. The source property key defines the input for the recorder, here I

use the vector recorder, which records the input values and accompanying timestamps as an output vector. I also use the min and max result filters, which record the maximum and minimum value of the results received, this is the reason why I use the sample-hold interpolation mode. To view a complete list of the result filters available I can enter the following in the terminal:

```
/omnetpp-5.5.1/MyWorkspace/test01/simulations$ ./run -h resultfilters
```

I can replace “resultfilters” with “resultrecorders” to get a complete list of the available result recorders. For further reading about property keys, filters and recorders, see subsections 4.14.2.2 and 4.15.2.3 in [56].

The results are recorded as either vector (.vec) or scalar (.sca) files, and OMNeT++ provides a rule-based result analysis option. This in order to automate the result analysis process. An example of rule-based result analysis is shown below:

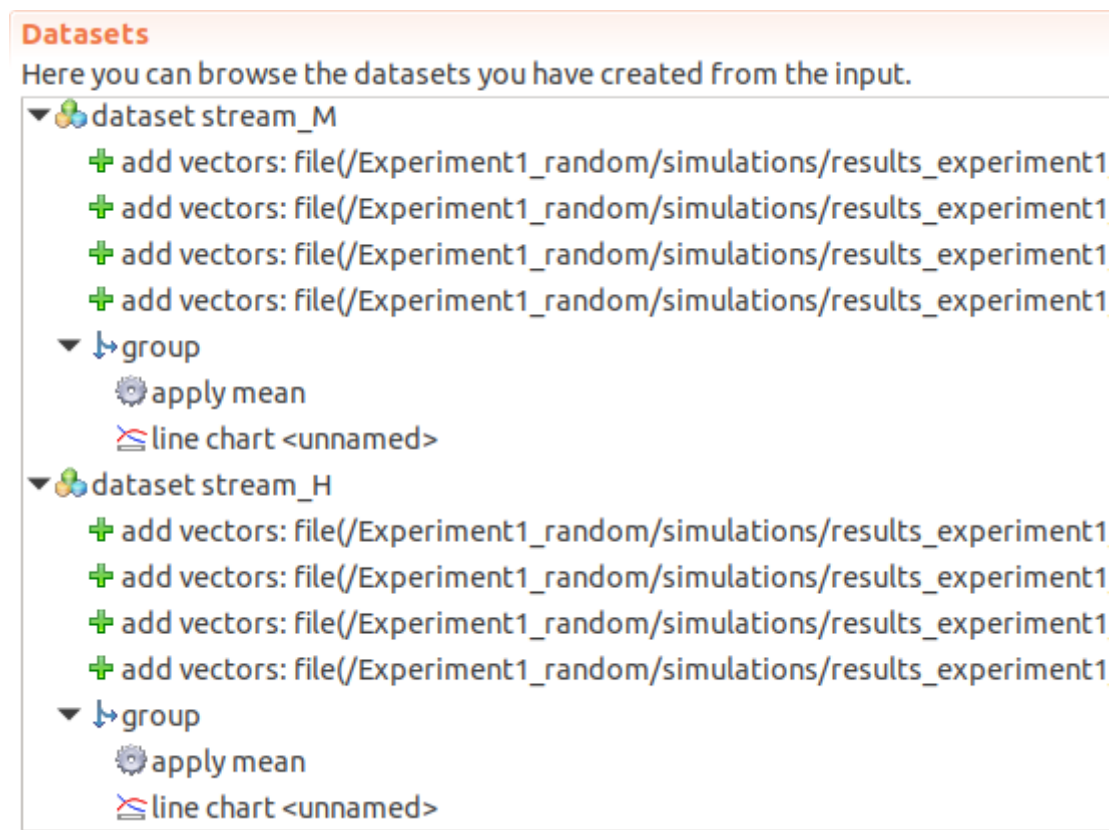


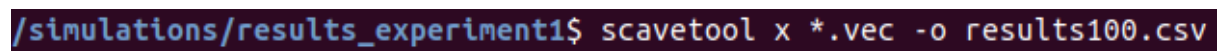
Figure 3.14: Rule-based result analysis.

Figure 3.14 shows two datasets created when running simulation experiments, if I change configuration parameters in the .ini file and run the simulation again, new line charts with the same computations (mean) will be generated.

The deadline constraint in [8] is imposed on a per frame basis; no frame in stream H or M should exceed the deadline, in the worst-case scenario, given the calculated bandwidth reservations according to Algorithm 1 and 2. Having familiarized myself with OMNeT++ and the analysis tool therein, I found it hard to analyze and perform computations on a per frame basis. According to the OMNeT++ documentation found here [59]: “The Analysis Tool in the OMNeT++ IDE is best suited for casual exploration of simulation results. If you are doing sophisticated result analysis, you will notice after a while that you have outgrown the IDE. The need for customized charts, the necessity of multi-step computations to produce chart input, or the sheer volume of raw simulation results might all be causes to make you look for something else.” [59] recommends Python for more sophisticated result analysis, unless one has experience with R or MATLAB, which I do not, in addition, Python is known for its easy syntax and minimalism[60]. I have therefore chosen to use Python for result analysis in the cases where I gather result data for each frame and then perform computations on the combined data. But I have also used the OMNeT++ result analysis tool in cases where it has been the more practical choice. Below, I will briefly show how some of the result analysis using Python was performed.

3.6 Result analysis using Python

I followed partially the Result Analysis with Python guide from [59]. This required me to install Jupyter Notebook , Python version 3, and the three Python packages: NumPy, Pandas, and Matplotlib. I then use the OMNeT++ scavetool in to convert the .vec and/or .sca files(s) to CSV format in order to read them into a DataFrame in Jupyter Notebook, using the `pandas.read_csv` function.



```
/simulations/results_experiment1$ scavetool x *.vec -o results100.csv
```

Figure 3.15: Using OMNeT++’s scavetool to convert a .vec result file to a .csv file.

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
%matplotlib inline

csv_string = "results100.csv"
data = pd.read_csv(csv_string)
payload = re.findall(r'\d+', csv_string)
transmission_time = ((int(payload[0]) + 42)*8 / 100000000)*1e6

dataframe_q4 = data[(data.name=='queueingTime:vector') & (data.module=='network1.switchA.eth[9].queue.queues[4]')]
dataframe_q5 = data[(data.name=='queueingTime:vector') & (data.module=='network1.switchA.eth[9].queue.queues[5]')]
dataframe_q6 = data[(data.name=='queueingTime:vector') & (data.module=='network1.switchA.eth[9].queue.queues[6]')]

qtime_dataframe_q4 = dataframe_q4['vecvalue'].dropna()
qtime_dataframe_q5 = dataframe_q5['vecvalue'].dropna()
qtime_dataframe_q6 = dataframe_q6['vecvalue'].dropna()

def createVector(w):
    w = w.reset_index()
    w = w.iloc[0,1]
    d = w.strip().split(' ')
    data=[1e6*float(d[i]) for i in range(0,len(d))]
    return data

qtime_vector_q4 = createVector(qtime_dataframe_q4)
qtime_vector_q5 = createVector(qtime_dataframe_q5)
qtime_vector_q6 = createVector(qtime_dataframe_q6)

def findMax(qtime_list):
    qtime_max = 0
    for i in range(0,len(qtime_list)):
        if qtime_list[i] > qtime_max:
            qtime_max = qtime_list[i]
    return qtime_max

def findAverage(qtime_list):
    qtime_total = 0
    for i in range(0,len(qtime_list)):
        qtime_total += qtime_list[i]
    return qtime_total / len(qtime_list)

qtime_max_q4 = findMax(qtime_vector_q4)
qtime_max_q5 = findMax(qtime_vector_q5)
qtime_max_q6 = findMax(qtime_vector_q6)

print(qtime_max_q4)
print(qtime_max_q5)
print(qtime_max_q6)

117.56
649.92
873.32

print(qtime_max_q4 + 123,36)
print(qtime_max_q5 + transmission_time)
print(qtime_max_q6 + transmission_time)

240.56 36
661.28
884.6800000000001

average_qtime_q4 = findAverage(qtime_vector_q4)
average_qtime_q5 = findAverage(qtime_vector_q5)
average_qtime_q6 = findAverage(qtime_vector_q6)

print(average_qtime_q4)
print(average_qtime_q5)
print(average_qtime_q6)

117.442439999999848
352.03999999999996
406.19962999999797

```

Figure 3.16: Example of how I have used Jupyter Notebook to analyze result files from OMNeT++

Chapter 4

Single node analysis

The work in this chapter is based on Experiment 1 from [8], and their presented bandwidth reservation algorithm. In this chapter I will use the bandwidth reservation algorithm to calculate the bandwidth reservation values for streams H and M for Experiment 1. I will then verify that the calculated bandwidth reservation values allows both stream H and M to meet the deadline requirement when the worst-case scheduling scenario occurs at the output port of the switch. The verification will first be done through calculations, I will then implement Experiment 1 into my chosen simulation model and verify that I get the same worst-case delay as in my calculations. Following this I will change the assumption made in [8] about the send interval of each source being constant, and evaluate how this change affects the maximum and average delay for the streams at the output port of the switch. A discussion about the feasibility of always having SR class A and B completely synchronized to the rest of the network, with no variation, will be discussed later in this chapter. I want to emphasize that when I refer to “Experiment 1”, I am referring to Experiment 1 from [8], as to avoid any possible confusion.

4.1 Bandwidth calculations

I will here present the calculated bandwidth reservation values for Experiment 1: varying the utilization via the payload, from [8], using their presented bandwidth reservation algorithms: Algorithm 1 and Algorithm 2. For a detailed description of how these calculations were performed, see Appendix B.

Payload	Eq. (7) (utilization constraint)	Eq. (8) (deadline constraint)
100 B	$\alpha^+_H \geq 4,544 \text{ Mb/s}$	$\alpha^+_H \geq \sim 3,938 \text{ Mb/s}$
200 B	$\alpha^+_H \geq 7,744 \text{ Mb/s}$	$\alpha^+_H \geq \sim 6,774 \text{ Mb/s}$
300 B	$\alpha^+_H \geq 10,944 \text{ Mb/s}$	$\alpha^+_H \geq \sim 9,664 \text{ Mb/s}$

400 B	$\alpha^+_H \geq 14,144 \text{ Mb/s}$	$\alpha^+_H \geq \sim 12,609 \text{ Mb/s}$
500 B	$\alpha^+_H \geq 17,344 \text{ Mb/s}$	$\alpha^+_H \geq \sim 15,610 \text{ Mb/s}$
600 B	$\alpha^+_H \geq 20,544 \text{ Mb/s}$	$\alpha^+_H \geq \sim 18,670 \text{ Mb/s}$
700 B	$\alpha^+_H \geq 23,744 \text{ Mb/s}$	$\alpha^+_H \geq \sim 21,789 \text{ Mb/s}$
800 B	$\alpha^+_H \geq 26,944 \text{ Mb/s}$	$\alpha^+_H \geq \sim 24,970 \text{ Mb/s}$
900 B	$\alpha^+_H \geq 30,144 \text{ Mb/s}$	$\alpha^+_H \geq \sim 28,214 \text{ Mb/s}$
1000 B	$\alpha^+_H \geq 33,344 \text{ Mb/s}$	$\alpha^+_H \geq \sim 31,524 \text{ Mb/s}$
1100 B	$\alpha^+_H \geq 36,544 \text{ Mb/s}$	$\alpha^+_H \geq \sim 34,902 \text{ Mb/s}$
1200 B	$\alpha^+_H \geq 39,744 \text{ Mb/s}$	$\alpha^+_H \geq \sim 38,349 \text{ Mb/s}$
1300 B	$\alpha^+_H \geq 42,944 \text{ Mb/s}$	$\alpha^+_H \geq \sim 41,867 \text{ Mb/s}$

Table 4.1: Calculated bandwidth reservation values for stream H, according to Algorithm 1 in [8].

Payload	Eq. (10) (utilization constraint)	Eq. (11) (deadline constraint)
100 B	$\alpha^+_M \geq 4,544 \text{ Mb/s}$	$\alpha^+_M \geq \sim 4,018 \text{ Mb/s}$
200 B	$\alpha^+_M \geq 7,744 \text{ Mb/s}$	$\alpha^+_M \geq \sim 7,018 \text{ Mb/s}$
300 B	$\alpha^+_M \geq 10,944 \text{ Mb/s}$	$\alpha^+_M \geq \sim 10,174 \text{ Mb/s}$
400 B	$\alpha^+_M \geq 14,144 \text{ Mb/s}$	$\alpha^+_M \geq \sim 13,503 \text{ Mb/s}$
500 B	$\alpha^+_M \geq 17,344 \text{ Mb/s}$	$\alpha^+_M \geq \sim 17,025 \text{ Mb/s}$
600 B	$\alpha^+_M \geq 20,544 \text{ Mb/s}$	$\alpha^+_M \geq 20,597966672682428 \text{ Mb/s}$
700 B	$\alpha^+_M \geq 23,744 \text{ Mb/s}$	$\alpha^+_M \geq 24,750211602634574 \text{ Mb/s.}$
800 B	$\alpha^+_M \geq 26,944 \text{ Mb/s}$	$\alpha^+_M \geq 29,01684751939554 \text{ Mb/s}$
900 B	$\alpha^+_M \geq 30,144 \text{ Mb/s}$	$\alpha^+_M \geq 33,608448493209333 \text{ Mb/s}$

1000 B	$\alpha_M^+ \geq 33,344 \text{ Mb/s}$	$\alpha_M^+ \geq 38,580089785717592 \text{ Mb/s}$
1100 B	$\alpha_M^+ \geq 36,544 \text{ Mb/s}$	$\alpha_M^+ \geq 44,002196256730853 \text{ Mb/s}$
1200 B	$\alpha_M^+ \geq 39,744 \text{ Mb/s}$	$\alpha_M^+ \geq 49,967060764089695 \text{ Mb/s}$
1300 B	$\alpha_M^+ \geq 42,944 \text{ Mb/s}$	$\alpha_M^+ \geq 56,597507165186769 \text{ Mb/s}$

Table 4.2: Calculated bandwidth reservation values for stream M, according to Algorithm 2 in [8].

Note that the total frame size at each payload is + 42 B, so at a payload of 100 B, the frame size is 142 B.

4.2 Worst-case calculations

Using the bandwidth reservation values presented in section 4.1, I will in this section calculate the worst-case delay for stream H and M at the output port of the switch for Experiment 1.

The calculations will be shown for a frame size of 142 B, and the results of the calculations for the remaining frame sizes will be listed. The calculation methods used in this section were introduced in subsection 2.2.1.1 in this thesis.

[8], which the work in this thesis is based on, references [61] for the worst-case interference by either a higher or a lower priority stream, and [62] for the worst-case interference caused by both higher and lower priority. For stream H the worst-case scheduling scenario, in the interfered case, is simply one lower priority frame. Since the source-set (stream H and M) is only schedulable up until a frame size of 1342 B, stream L will always have the larger frame size, compared to stream M. This means that the worst-case delay for a stream H frame is having to wait for one L frame, compared to the un-interfered case. For stream M, [62] provides a figure showing a generic worst-case scenario for a stream M interfered by both higher and lower priority, as shown below:

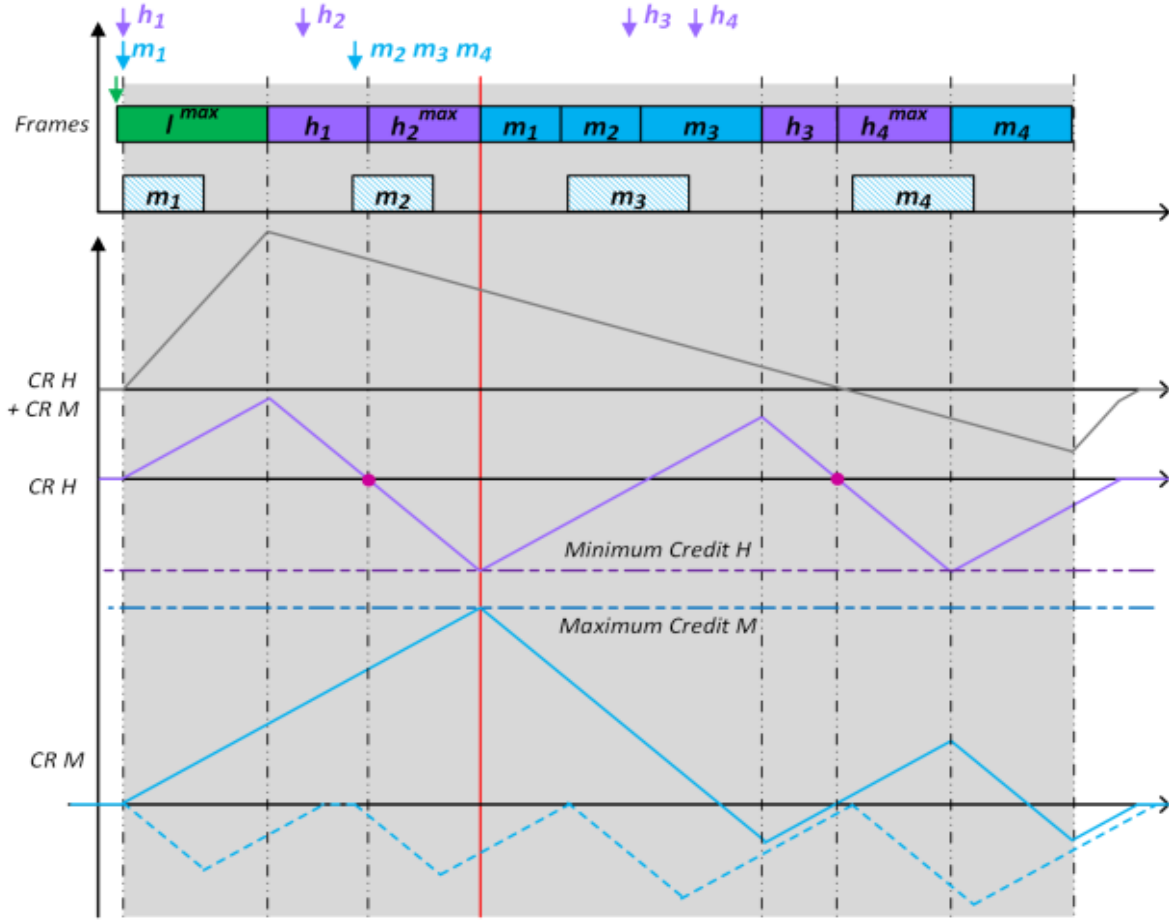


Figure 4.1: Generic construction of worst-case for stream M. Obtained from [62].

The worst-case scenario for a stream M frame is linked to the maximum credit value. In Figure 4.1 the maximum credit value occurs when there arrives a maximum sized lower-priority frame immediately before the medium-priority frame experiencing the maximum delay, m_1 , together with a high-priority frame, h_1 . After the lower-priority frame is forwarded, h_1 is forwarded, since the credit value of stream H is positive, and it has the higher priority. Due to the combination of bandwidth reservation for stream H and the size of frame h_1 , the h_1 frame forward takes the credit value for stream H down to zero. During the forward of frame h_1 there arrives a maximum sized H frame, and since the credit value of stream H is zero after the forward of frame h_1 , the maximum sized stream H frame is forwarded. After this, stream M has reached its maximum credit value, and frame m_1 has experienced its maximum delay, compared to the un-interfered schedule.

I first calculate the maximum credit value boundary, $hiCredit$, as specificized in IEEE Std 802.1Q ([19]) for each frame size in Experiment 1, using the calculation method as shown in subsection 2.2.1.1 in this thesis. The reason for this is to ensure that I am not limited by the $hiCredit$ value when calculating the worst-case; I might calculate the class M credit going

from 0 to X while waiting for an interfering L frame, but the remainder of the calculations will be wrong if hiCredit limits the credit value to only reach X-y. It is worth noting that the credit-based shaping algorithm in NeSTiNg does not take into account hiCredit (or loCredit), but this does not matter when looking at the worst-case scheduling scenario for Experiment 1, because based on my calculations, hiCredit (or loCredit) is never reached. I show the calculations as performed for a frame size of 142 B with the corresponding bandwidth reservation, as shown in the previous section, for stream M.

At a frame size of 142 B the idleSlope is equal to the bandwidth reservation, which is 4,544 Mb/s, as seen in Table 4.2. The port transmit rate is equal to the link speed, which is 100 Mb/s.

$$\text{hiCredit} = \text{maxInterferenceSize} \times (\text{idleSlope} / \text{portTransmitRate})$$

$$\text{hiCredit} = 16000 \times (4544000 / 100000000) = 16000 \times 0,04544 = \underline{727,04}$$

$$\text{sendSlope} = \text{idleSlope} - \text{link speed} = 4,544 \text{ Mb/s} - 100 \text{ Mb/s} = -95,456 \text{ Mb/s}$$

$\text{maxFrameSize} = 100000000 \times 0,00025 \times 0,04544 = 1136 \text{ bit}$ (note that I am using a class measurement interval of 250 μs , corresponding to SR class B, as this will correspond to stream M, and SR class A will correspond to stream H).

$$\text{loCredit} = \text{maxFrameSize} \times (\text{sendSlope} / \text{portTransmitRate})$$

$$\text{loCredit} = 1136 \times (95456000 / 100000000) = 1136 \times 0,95456 = \underline{1084,38016}$$

In order to construct the correct worst-case scenario for stream M, I also have to figure out if I can forward two stream H frames consecutively after waiting for one interfering L frame, according to the parameters in Experiment 1. The calculation for this will be shown below, for a frame size of 142 B, the same calculation is performed at each frame size in order to ensure that I construct the correct worst-case scheduling scenario for stream M.

The transmission time for an L frame, C_L , is $0,00012336 \text{ s} = 123,36 \mu\text{s}$. During the L transmission, assuming the credit of stream H is at zero, the credit for stream H will then increase from zero to: $\text{idleSlope} \times C_L = 4544000 \times 0,00012336 = 560,54784$. Following this, a stream H frame will be forwarded from the switch. The transmission time for a stream H frame, C_H , at this payload, is $0,00001136 \text{ s} = 11,36 \mu\text{s}$. The cost of forwarding a stream H frame will then be: $\text{sendSlope} \times C_H = 95456000 \times 0,00001136 = 1084,38016$. The credit will go from 560,54784 to -523,83232, and I cannot forward another H frame.

Below I will show the worst-case calculations performed for stream H and M, at a frame size of 142 B (payload = 100 B). Note that I use an interframe spacing, according to IEEE Std

802.3 ([63], from table 4-2), that is, an idle time between two consecutive frame forwards. This idle time “is intended to provide interpacket recovery time for other CSMA/CD sublayers and for the physical medium” [63]. In the case of a 100 Mb/s link, this time equals 96 bits, meaning the time it takes to transmit 96 bits over a 100 Mb/s link, which equals 0,96 μ s, this idle time is also used in NeSTiNg. It is important to note when the interframe spacing is used and its value, when looking at the calculations shown below. Also, according to IEEE Std 802.1Q ([19]), the credit does not start to increase back to zero until the frame forward is completed.

The “dep” event marks when a frame has finished its forward across the output port of the switch. The “arr” event marks the time when one or multiple frames arrive in the queue at the output port of the switch.

Event	Time [μ s]	Buffer	Delay [μ s]	Credit value		
h1,h2,h3,h4 arr	0	h1,h2,h3,h4		0		
h4 dep	11,36	h1,h2,h3	11,36	-1084,38016		
cr = 0	250			0		
h3 dep	261,36	h1,h2	261,36	-1084,38016		
cr = 0	500			0		
h2 dep	511,36	h1	511,36	-1084,38016	C_H	11,36 μ s
L arr	749				C_L	123,36 μ s
cr = 0	750			0	Idle time	0,96 μ s
L dep	872,36			556,00384	idleSlope	4544000 b
h1 dep	884,68		884,68	-524,01408	sendSlope	-95456000 b

Figure 4.2: Worst-case scheduling scenario for stream H at a frame size of 142 B.

The worst-case un-interfered scenario is all four stream H frames arriving at the same time. And the interfered scenario is the arrival of an L frame right before a stream H frame can transmit, in this case I show this by having an L frame arrive 1 μ s before the credit reaches zero. In these calculations I assume no time is needed between a frame arrival and starting to forward that same frame from the switch. Combining the worst-case-un-interfered scenario with the worst-case interfered scenario I get the scheduling scenario shown in Figure 4.2.

Using the same example, but placing the L frame at the beginning, I show that interfering traffic has an acute effect, due to the credit generating being able to generate above zero when waiting for interfering traffic.

Event	Time [μ s]	Buffer	Delay [μ s]	Credit value
L arr	0			0
h1,h2,h3,h4 arr	1	h1,h2,h3,h4		0
L dep	123,36			556,00384
h4 dep	135,68	h1,h2,h3	134,68	-524,01408
cr = 0	251			0
h3 dep	262,36	h1,h2	262,36	-1084,38016
cr = 0	501			0
h2 dep	512,36	h1	511,36	-1084,38016
cr = 0	751			0
h1 dep	762,36		761,36	-1084,38016

Figure 4.3: Illustrating the effect of interfering traffic on a credit-based shaper traffic class.

Comparing to the scheduling scenario in figure 4.2: we can see that the interfering L frame affects mostly the first frame having to wait, h4, with 123,32 μ s. The second frame, h3, is only delayed by 1 μ s. The third frame, h2, is not affected at all.

Below I show the worst-case scheduling scenario for stream M:

Event	Time [μs]	Buffer	Delay [μs]	Credit value		
m1,m2,m3,m4 arr	0	m1,m2,m3,m4		0		
m4 dep	11,36	m1,m2,m3	11,36	-1084,38016		
cr = 0	250			0		
m3 dep	261,36	m1,m2	261,36	-1084,38016		
cr = 0	500			0		
m2 dep	511,36	m1	511,36	-1084,38016		
L arr	749				C_M	11,36 μs
cr = 0	750			0	C_L	123,36 μs
L dep	872,36			556,00384	Idle time	0,96 μs
h dep	884,68			611,98592	idleSlope	4544000 b
m1 dep	897		897	-468,032	sendSlope	-95456000 b

Figure 4.4: Worst-case scheduling scenario for stream M at a frame size of 142 B.

The worst-case un-interfered scenario is all four stream M frames arriving at the same time. The worst-case interfered scenario is when a stream L frame arrives just before the credit of stream M becomes zero, and after the stream L frame forward is finished, a stream H frame is available and forwarded. As long as the stream M frames arrive at the same time, and the worst-case interfered scenario occurs for the last M frame, it does not matter if any H frames arrive in between. The reason for this is that if other stream H frames arrived in between the two aforementioned points, they would only acutely affect the stream M departures. For order's sake I provide an example below to show this:

Event	Time [μs]	Buffer M	Buffer H	Delay [μs]	Credit value M	Credit value H
m1,m2,m3,m4 arr	0	m1,m2,m3,m4			0	
h1,h2,h3,h4 arr	0		h1,h2,h3,h4			0
h4 dep	11,36		h1,h2,h3	11,36	51,61984	-1 084,38
m4 dep	23,68	m1,m2,m3		23,68	-1028,39808	
cr _H = 0	250					0
cr _M = 0	250				0	
h3 dep	261,36		h1,h2	261,36	51,61984	-1 084,38
m3 dep	273,68	m1,m2		273,68	-1028,39808	
cr _H = 0	500					0
cr _M = 0	500				0	
h2 dep	511,36		h1	511,36	51,61984	-1 084,38
m2 dep	523,68	m1		523,68	-1028,39808	
L arr	749					
cr _H = 0	750					0
cr _M = 0	750				0	
L dep	872,36				556,00384	556,00384
h1 dep	884,68			884,68	611,98592	-523,83216
m1 dep	897			897	-472,21232	

Figure 4.5: Worst-case scheduling scenario for stream M and H, at a frame size of 142 B.

In the example shown in Figure 4.5, the transmission time for a stream H frame is the same as the transmission time for a stream M frame, 11,36 μs. The idleSlope is the same for both streams, 4544000 bits. The sendSlope is the same for both streams, 95456000 bits. And, as in the previous calculation examples shown, I use an interframe spacing of 0,96 μs between two consecutive frame forwards.

The worst-case scheduling scenario calculations for stream H (Figure 4.2) and M (Figure 4.4) has now been shown for a frame size of 142 B. At this frame size both streams are within the deadline requirement of 1000 μs. Note that while Figure 4.5 shows the worst-case for both stream H and M, this is due to the credit for both streams reaching zero at the same times. For a frame size of 642 B and above, the credit of stream M will reach zero before the credit value of stream H, due to the extra reserved bandwidth for stream M, and the exact scheduling scenario shown in Figure 4.5 would then not provide the worst-case delay for both streams.

The worst-case calculations, as shown in Figure 4.4 and Figure 4.2, has been performed for each schedulable frame size, using the calculated bandwidth reservation values as shown in Table 4.1 and 4.2, and are listed in the table below

Frame size	Calculated worst-case delay stream H [μs]	Calculated worst-case delay stream M [μs]
142 B	884,68	897
242 B	892,68	913
342 B	900,68	929
442 B	908,68	945
542 B	916,68	961
642 B	924,68	975,035
742 B	932,68	962,508
842 B	940,68	955,423
942 B	948,68	947,687
1042 B	956,68	939,21
1142 B	964,68	929,878
1242 B	972,68	919,684
1342 B	980,68	908,07

Table 4.3: Calculated worst-case delays for stream H and M.

At all frame sizes, both streams are within the deadline requirement. The reason for the reduction of delay for stream M, at a frame size of 642 B and above, is due to the increased bandwidth reservation (deadline constraint), compared to stream H.

4.3 Worst-case simulation

In this section I will describe how I implemented Experiment 1 into my chosen simulation model. I will then show the results of simulating the worst-case scheduling scenario, as introduced in the previous section, for each frame size, using the calculated bandwidth reservation values from Tables 4.1 and 4.2.

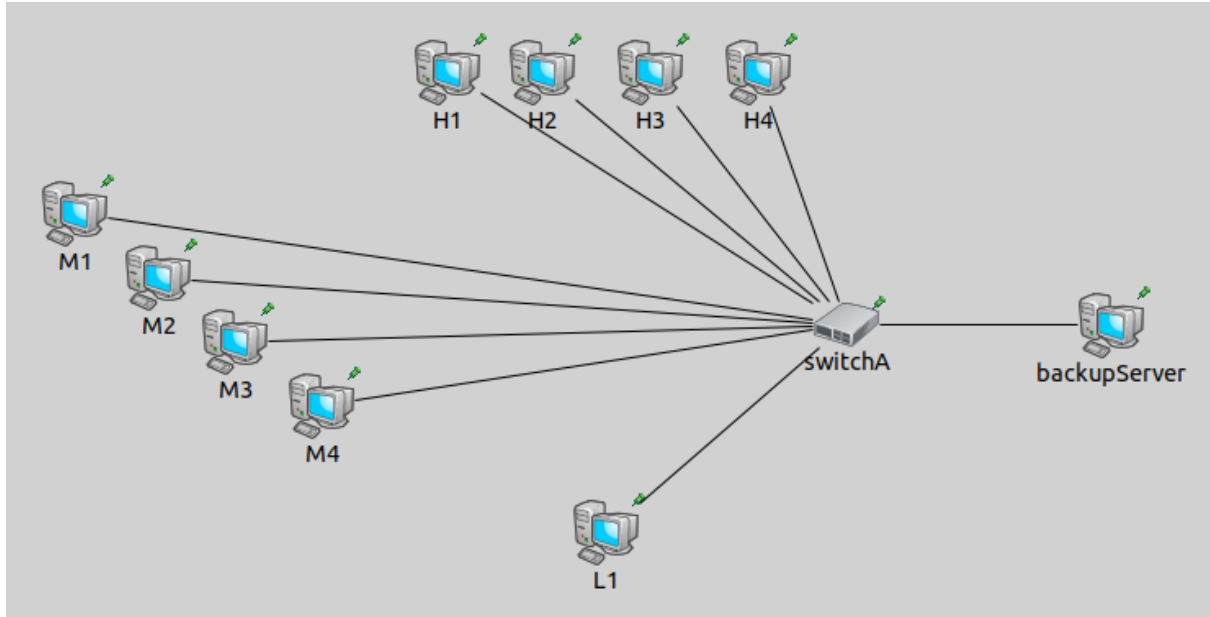


Figure 4.6: Topology of Experiment 1 from [8], as implemented into OMNeT++/INET/NeSTiNg.

All sources and the backup server are of type `VlanEtherHostQ`, the switch is of type `VlanEtherSwitchPreemptable`. All sources send their frames to the backup server. The important part is how much time each frame spends in its assigned output queue + the time it takes to transmit the frame across the output port of the switch, which cannot exceed the deadline of $1000\ \mu\text{s}$. Sources H1-H4 represents the four high priority sources, they are all assigned a PCP value of 6, and thereby assigned to queue 6, corresponding to SR class A. Sources M1-M4 represents the four medium-priority sources, they are all assigned a PCP value of 5, and thereby assigned to queue 5, corresponding to SR class B. Source L1 sends low priority traffic using a PCP value of 4. The link speed is set to 100 Mb/s, and the period of each source is set to $1000\ \mu\text{s}$, according to the parameters in Experiment 1. I use Modification 1 and Modification 2, shown in section 3.4 of this thesis. I modify the start time parameter for each source in the .ini file, in order to construct the worst-case scheduling scenario for stream H, and stream M, at each frame size. The worst-case delay, as recorded when running the simulations, are listed in the table below.

Frame size	WC delay stream H [μ s]	WC delay stream M [μ s]	Average delay WC-H [μ s]	Average delay WC-M [μ s]
142 B	884,68	897	417,19	420,27
242 B	892,68	913	425,19	430,27
342 B	900,68	929	433,19	440,27
442 B	908,68	945	441,19	450,27
542 B	916,68	961	449,19	460,27
642 B	924,68	975,035	457,19	469,287
742 B	932,68	962,508	465,19	465,024
842 B	940,68	955,423	473,19	463,481
942 B	948,68	947,687	481,19	461,613
1042 B	956,68	939,21	489,19	459,51
1142 B	964,68	929,878	497,19	456,709
1242 B	972,68	919,684	505,19	453,546
1342 B	980,68	908,07	513,19	449,805

Table 4.4: Worst-case delay and average delay for streams H and M, as recorded when running worst-case scheduling scenarios.

As we can see from Table 4.4, I record the same maximum delay values, at each frame size, as the calculated ones (Table 4.3). I also provide the average delay times for the stream experiencing the worst-case scheduling scenario, using the exact scenarios shown in Figures 4.2 and 4.4. Due to the gradual increase in bandwidth reservation for stream M, at a frame size of 642 B and above, the credit generation rate increases, which also means that the cost of forwarding a frame decreases, resulting in the credit value of stream M recovering to zero faster after a frame transmission, for each increase in frame size. The increase in frame size is not enough to offset this, translating to shorter maximum delay and average delay times for stream M at a frame size of 642 B and above.

4.4 Discussing the feasibility of accurate time synchronization

The bandwidth reservation analysis provided in [8] assumes a constant period for each source, meaning the sources are completely synchronized with the rest of the network, this might not always be feasible. [14] mentions the possibility of misbehaving talkers (sources) transmitting at rates higher than what has been assigned/reserved. [64] provides an extensive overview of different factors which can contribute to a deterioration, or in some cases, a complete loss of synchronization between devices in distributed systems. [64] puts special emphasis on the Precision Time Protocol (PTP), as defined in IEEE 1588. This is relevant because AVB, wherein the CBS is defined, uses the clock synchronization standard IEEE 802.1AS, which again uses the PTP [14]. I must also mention that [8] talks about the CBS in the context of TSN. TSN uses a revised version of IEEE 802.1AS: IEEE 802.1AS-Rev. This revised version provides increased fault-tolerance and a seamless low latency handover and recovery of time synchronization in failure nodes [25]. [64] still applies, and I will list the main factors it mentions that may affect time synchronization:

- Clock synchronization over Ethernet uses a message-based approach, this means dedicated packages for this purpose are exchanged regularly between devices. One of the errors which might occur during this message exchange is an inaccuracy in the timestamp inserted in the package by the CPU of the sender: there might be a delay from the time the CPU inserts the timestamp until the message is actually sent; first the packet will be moved from the main memory to its local buffer, here it might be delayed further by interfering traffic. “The time span between drawing a time stamp from the local clock and sending the corresponding packet via the physical channel may vary significantly because it encompasses a number of tasks each of which could be interrupted for an unknown amount of time by other processes with higher priority within any nonreal-time operating system” [64]. In addition, at the reception of clock synchronization messages, the reception time needs to be noted as accurately as possible, this process is also susceptible to similar inaccuracies.

- Variations in packet processing and forwarding time, also known as packet delay variation (PDV), introduces the possibility of network related time inaccuracies. A packet might arrive at a switch and be forwarded by the switch fabric hardware unit to a specific egress port, as decided by the active routing rules and the header information found in the packet, here the packet might experience an empty queue meaning it can be forwarded immediately or it might have to wait for other packets. “The amount of PDV a specific device adds to a packet is not only dependent on the network load that the switch has to process at that time but also on its underlying hardware architecture” [64]. Expedite forwarding where the packet is forwarded immediately after interfering frame, thereby skipping the line, is mentioned as a mitigation of PDV but it cannot completely remove the problem.
- There is also the possibility of synchronization messages being lost due to link failures or malicious attacks.

4.5 Simulating variations in send interval

In this section I start making modifications to the period (send interval) and start time of the sources in Experiment 1, I do this by using Modification 3, as shown in section 3.4 of this thesis. The start times for all experiment are generated in the range [0-1000] μ s and are shown below, the send interval ranges are specified for each simulation experiment. I also use the calculated bandwidth reservation values, as shown in Tables 4.1 and 4.2, apart from this, all other parameters are the same as in Experiment 1. Note that I am using a simulation time of 1 second when running the simulation experiments, unless otherwise specified. During this simulation time each source sends 1000 frames, assuming a period of 1000 μ s, which I show is enough to see how the send interval variations affect the queue times of stream H and M. Running the simulation experiments longer did not provide any additional benefits for the purpose of my analysis, hence using the default simulation time of 1 second.

Also, note that I do not list the stream L queue times. The reasons for this are:

1. It will be addressed in a later section, and I want to highlight streams H and M.
2. There are no meaningful observations to comment on, yet, when it comes to variations in stream L queue times due to it being affected by stream H and M traffic; the utilization of streams H and M, at a frame size of 1342 B, assuming a send interval

of 1ms, is 42,944 Mb/s, each. This leaves 14,112 Mb/s for stream L, which has a utilization of 12,336 Mb/s. Meaning, as the utilization, and thereby bandwidth reservation for streams H and M increases, the time window where queue 4 (stream L) can forward a frame decreases, and the average queue times for stream L thereby increases. Small variations for stream L queue times are observed due to different start times and send interval averages, when using different seed-sets.

H1	H2	H3	H4	M1	M2	M3	M4	L1
684 μ s	629 μ s	192 μ s	835 μ s	763 μ s	707 μ s	359 μ s	9 μ s	723 μ s

Table 4.5: Start times generated for each source using the default seed-set and range.

H1	H2	H3	H4	M1	M2	M3	M4	L1
37 μ s	908 μ s	72 μ s	767 μ s	905 μ s	715 μ s	645 μ s	847 μ s	960 μ s

Table 4.6: Start times generated for each source using the seed-set 1.

4.5.1 Sim 1: $\pm 10\%$ variation in send interval

In this simulation experiment each source has a variable send interval in the range [900,1100] μ s, that is, $1000 \mu\text{s} \pm 10\%$. This should result in an average send interval of 1000 μ s, which is the send interval (period) used when calculating the bandwidth reservation values.

H1	H2	H3	H4	M1	M2	M3	M4	L1
1001,64 128256 μ s	999,399 μ s	998,6686 6267 μ s	998,2257 7422 μ s	1001,52 805611 μ s	1000, 224 μ s	998,695 30569 μ s	1002,71 042084 μ s	1002,04 909819 μ s

Table 4.7: Send interval average for each source, using default seed-set.

The average send interval for stream H (all H sources) is 999,483 μ s, and for stream M the average send interval is 1000,789 μ s. Below I list the maximum delay for streams H and M,

and average delay times, as recorded when running the simulation for each frame size, using a send interval variation of $1000 \pm 10\%$.

Frame size	Maximum delay H [μ s]	Maximum delay M [μ s]	Average delay H [μ s]	Average delay M [μ s]
142 B	1492,67	1221,67	710,48	528,15
242 B	1492,67	1237,67	719,08	539,91
342 B	1544,67	1253,67	727,77	552,1
442 B	1544,67 μ s	1269,67	736,39	564,49
542 B	1544,67	1285,68	745,2	578,09
642 B	1544,67	938,06	755,94	402,61
742 B	1557,65	844,0	765,22	233,51
842 B	1544,68	783,68	775,88	227,33
942 B	1559,32	783,68	786,7	230,23
1042 B	1583,32	783,68	799,31	235,42
1142 B	1607,32	783,68	813,92	244,32
1242 B	1631,32	823,28	829,47	252,2
1342 B	1583,53	850,92	851,69	293,21

Table 4.8: Maximum delay and average delay times for Sim 1. Using a send interval variation of $1000 \pm 10\%$, with default seed-set.

As shown in the Table 4.8, stream H breaches the deadline at all frame sizes. Stream M breaches the deadline up to and including a frame size of 542 B. At a frame size of 642 B and above, the bandwidth reservation for stream M follows the deadline constraint and therefore has an increased bandwidth reservation, compared to the same frame sizes for stream H.

An interesting observation here is that, even though the send-interval average for stream M is above 1000 μ s, stream M still breaches the deadline up to and including a frame size of 542 B. Note this observation when looking at the rest of the simulation experiment data below. Also, this is not something only happening to stream M, since the experiments below does not show this occurring for stream H: running Sim 1 with seed-set 1 causes stream H to experience a send interval average of 1000,438 μ s. With this send interval average stream H

breaches the deadline at all frame sizes. Simulation data for Sim 1 using seed-set 1 is provided in Appendix D.

In Appendix C, I provide deadline breach event logs for both stream H and M, detailing the events occurring at the output port of the switch, during the time interval when the first frame of stream H and the first frame of stream M breaches the deadline. For each event log I provide an accompanying graph, showing the send interval mean for each source in the stream, during the time interval of the deadline breach.

4.5.2 Sim 2: $\pm 5\%$ variation in send interval

In this simulation experiment I reduce send interval variation range to $[950, 1050]$ μs , in order to see how a reduction in variation affects maximum delay and average delay times. Using the random function (Modification 3 in section 3.4) with a new send interval range results in new values being generated. The start times are still the same since I did not change start time range.

H1	H2	H3	H4	M1	M2	M3	M4	L1
999,802 μs	998,91508 491 μs	998,79041 916 μs	999,0649 3506 μs	1000,474 47447 μs	999,94 μs	999,4755 2447 μs	1001,045 04504 μs	1000,205 μs

Table 4.9: Send interval average for each source, using default seed-set.

The average send interval for stream H is 999,143109782 μs , and for stream M the average send interval is 1000,233760995 μs .

Frame size	Maximum delay H [μ s]	Maximum delay M [μ s]	Average delay H [μ s]	Average delay M [μ s]
142 B	1370,36	971,36	693,75	456,5
242 B	1378,36	979,36	702,04	465,17
342 B	1386,36	987,36	709,92	473,59
442 B	1394,36	995,36	718,36	482,13
542 B	1402,36	1003,36	726,8	490,7
642 B	1410,36	766,0	738,14	290,7
742 B	1418,36	681,86	747,21	203,38
842 B	1426,36	666,78	757,88	206,31
942 B	1559,32	783,68	786,7	230,23
1042 B	1506,68	689,68	782,87	218,95
1142 B	1522,67	694,68	798,01	228,86
1242 B	1538,67	728,6	815,05	237,22
1342 B	1554,67	795,24	833,93	269,25

Table 4.10: Maximum delay and average delay times for Sim 2. Using a send interval variation of $1000 \pm 5\%$, with default seed-set.

Compared Sim 1, a 5% reduction in send interval variation provides a reduction in maximum delay and average delay times, most notably for stream M up to a frame size of 642 B, but also afterwards, with the exception of 942 B. With this reduction in send interval variation, stream M is able to meet the deadline for all frame sizes apart from at a frame size of 542 B. Keep in mind that compared to Sim 1 the average send interval for stream H is actually slightly lower in this experiment, and the average send interval for stream M is slightly higher.

I also run Sim 2 with seed-set 1, which generates the following send interval averages for each source:

H1	H2	H3	H4	M1	M2	M3	M4	L1
1000,073 μs	1001,072 07207 μs	1001,052 05205 μs	1000,07 μs	999,244 μs	999,941 μs	1000,274 μs	999,617 μs	999,492 μs

Table 4.11: Send interval average for each source, using seed-set 1.

The average send interval for stream H is 1000,56678103 μs and for stream M the average send interval is 999,769 μs . Note that the start times are now different, compared to using the default seed-set.

Frame size	Maximum delay H [μs]	Maximum delay M [μs]	Average delay H [μs]	Average delay M [μs]
142 B	822,68	1155,0	338,27	575
242 B	822,68	1163,0	346,81	583,39
342 B	822,68	1171,0	355,3	592,03
442 B	822,68	1179,0	366,61	601,26
542 B	845,68	1187,0	378,2	610,61
642 B	845,68	810,68	382	290,53
742 B	853,65	750,86	392,92	219,09
842 B	873,78	735,78	403,66	221,35
942 B	891,64	720,04	414,93	225,38
1042 B	896,64	703,57	427,08	231,5
1142 B	922,28	731,64	441,94	241,31
1242 B	921,64	782,6	456,4	250,55
1342 B	936,56	797,36	476,27	274,39

Table 4.12: Maximum delay and average delay times for Sim 2. Using a send interval variation of $1000 \pm 5\%$, with seed-set 1.

With the new send interval times generated by seed-set 1, stream H experiences a significant reduction in maximum delay and average delay times, and is now within the deadline requirement at every frame size, compared to using the default seed-set. While stream M experiences an increase in maximum delay and average delay times.

Pointing out some observations so far:

- It seems that a reduction in send interval variation provides a reduction in maximum delay and average delay times, as observed when comparing Sim 1 (Table 4.8) to Sim 2 (Table 4.10).
- While at this point somewhat conflicting, it seems that an increase in send interval average for a stream causes a reduction in maximum delay and average delay times for said stream. And, that a decrease in send interval average for a stream causes an increase in maximum delay and average delay times for said stream. I say that these observations are somewhat conflicting because: when going from Sim 1 to Sim 2 default seed-set, there was a reduction in send interval average for stream H, but a decrease in maximum delay and average delay times, but also a reduction in send interval variation. While for stream M there was a slight increase in send interval average, and a decrease in maximum delay and average delay times. Going from Sim 2 default seed-set to Sim 2 seed-set 1, there was an increase in send interval average for stream H and a reduction in maximum delay and average delay times.
- At some frame sizes, even though the stream has a send interval average of 1000 μ s or above, the stream still breaches the deadline. A thing to consider here is that even though the average is at or above 1000 μ s, the stream still might have experienced time intervals where the send interval was well beneath 1000 μ s.

In Appendix C I provide a deadline breach event log for stream H (default-seed set), together with a send interval mean graph for the time period of the breach. I also provide an event log for stream M (default seed-set), together with a send interval mean graph. As this is an interesting comparison, seeing as stream H, having the higher priority, breaches the deadline, but not stream M, shown in Table 4.10. The conclusion from going over the event logs mentioned above: consistent with the send interval averages recorded for streams H and M, I observe that there occurs an accumulation of 5 stream H frames in queue 6, causing the fifth frame to breach the deadline. Whereas for stream M, the maximum number of frames stored in the queue, during the time period where the stream experiences its largest delay, is 3.

4.5.3 Sim 3: $\pm 2\%$ variation in send interval

I further reduce the send interval variation, the range is now [980,1020]. Which produces the following send interval averages:

H1	H2	H3	H4	M1	M2	M3	M4	L1
1000,412 41241 μs	1000,337 μs	999,3746 2537 μs	1000,627 62762 μs	1000,646 64664 μs	999,66 μs	1000,200 99999 μs	999,917 08291 μs	999,746 μs

Table 4.13: Send interval average for each source, using default seed-set.

The total average for stream H is 1000,18791635 μs , and for stream M 1000,106182385 μs .

Frame size	Maximum delay H [μs]	Maximum delay M [μs]	Average delay H [μs]	Average delay M [μs]
142 B	535,68	537,36	263,35	220,44
242 B	543,68	545,36	271,35	228,99
342 B	559,68	553,36	279,31	238,05
442 B	575,68	561,36	287,66	247,33
542 B	591,68	569,36	296,57	256,4
642 B	599,37	566,68	306,82	183,79
742 B	599,35	543,86	318,4	164,16
842 B	604,78	528,78	329,9	168,59
942 B	620,86	525	343	175,13
1042 B	645,75	559	356,32	185,42
1142 B	674,28	534,6	371,46	198,86
1242 B	722,28	590,6	387,25	207,26
1342 B	704,28 μs	624,6	404,23	229,96

Table 4.14: Maximum delay and average delay times for Sim 3. Using a send interval variation of $1000 \pm 2\%$, with default seed-set.

A further reduction in send interval variation range seems to provide further reductions in maximum delay and average delay times. If I compare the stream H data shown in Table 4.14

above, where stream H has a send interval average of 1000,18791635 μs and a $\pm 2\%$ variation, with the stream H data shown in table 4.12, where the stream H has an send interval average of 1000,56678103 μs and a $\pm 5\%$ variation. Even though the streams have a very similar send interval average, reducing the variation provides a significant reduction in maximum delay and average delay times.

I also provide data from running this simulation using seed-set 1, in Appendix D. And in Appendix C, I provide a maximum deadline event log for stream M (default seed-set) and stream H (seed-set 1), with accompanying send interval mean graphs.

Commenting on the pattern of a reduction in send interval variation seeming to provide a reduction in maximum delay and average delay times, as seen in all simulation experiments so far: when using a uniform pseudorandom number generator, there will inevitably be time periods where it generates more numbers from the lower end of the range. It seems that when there is a larger variation range, meaning a larger discrepancy between the middle (used to reserve the bandwidth) and the lower bound, this increases the likelihood of a deadline breach occurring, as it requires a shorter time period of lower values before a breach occurs, compared to using a smaller send interval variation range.

4.6 Further examining send interval variations

In this section I will look closer at how the send interval variation affects the delay times.

Below I am showing a graph depicting the variations in queue times for stream M at a frame size of 142 B from Sim 1 ($\pm 10\%$ variation), using the default seed-set.

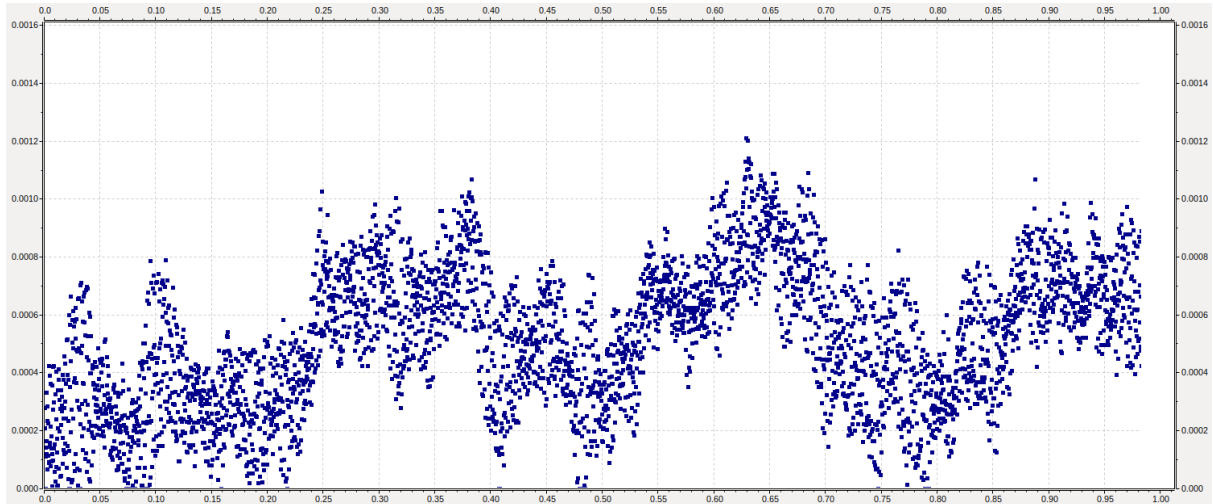


Figure 4.7: queue time variations for stream M at a frame size of 142 B, from Sim 1, using default seed-set. Axes are in seconds.

Zooming in on the first two spikes in queue times:

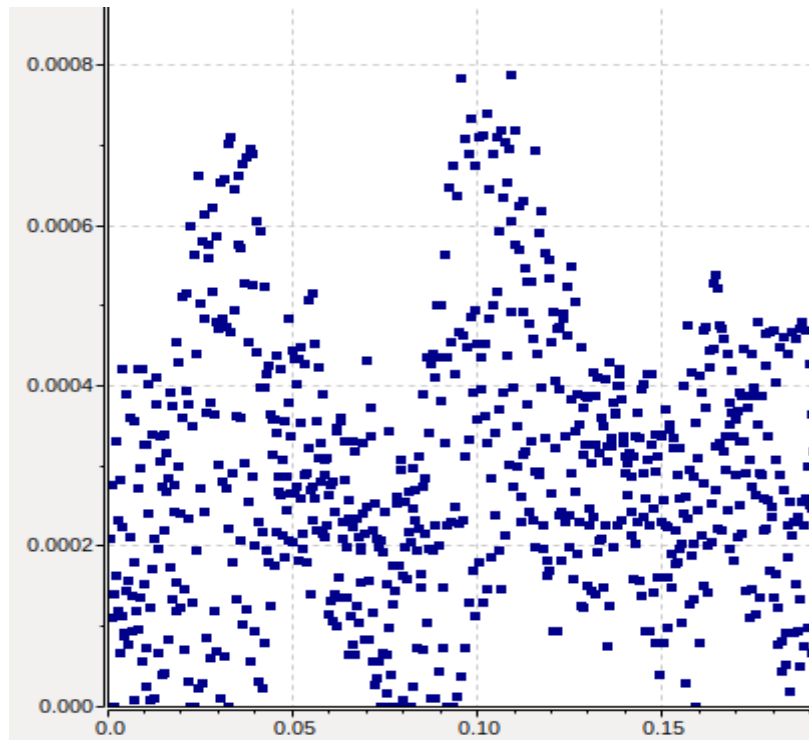


Figure 4.8: queue time variations for stream M at a frame size of 142 B, from Sim 1, using default seed-set. Axes are in seconds.

Each dot on the x-axis is a 10 ms increase, from left to right. At 18 ms we start to see an aggressive rise in the queue time for stream M, reaching a peak at 33 ms, and then decreasing and reaching a bottom at 66 ms, then starting to rise again at 85 ms. The average send interval for stream M for the first 15ms of the simulation time is 1010 μ s. The average send interval between 15-25 ms is 993 μ s, and between 25-35 ms the average is 990 μ s. Between 35-45 ms the average send interval is 1013 μ s. For both streams, at all frame sizes, I observe the same pattern: periods of lower send interval times result in higher queue times and vice versa.

In Appendix E I present a calculation example based on Sim 1, where the send interval variation range is [900,1100]. In each cycle, I have all four stream H frames arrive at the exact same time, I then vary the send interval for stream H: setting the send interval to the minimum (900 μ s), middle (1000 μ s), or the maximum (1100 μ s). In the example I am using a frame size of 142 B, and the calculated bandwidth reservation value for this frame size, as shown in Table 4.1. An excerpt from this example is shown below.

Event	Time [μ s]	Buffer	Delay [μ s]	Credit
h1,h2,h3,h4 arr	0	h1,h2,h3,h4		0
h4 dep	11,36	h1,h2,h3	11,36	-1084,38016
cr = 0	250			
h3 dep	261,36	h1,h2	261,36	-1084,38016
cr = 0	500			
h2 dep	511,36	h1	511,36	-1084,38016
L arr	749			
cr = 0	750			
L dep	873,32			560,36608
h1 dep	884,68		884,68	-524,01408
h1,h2,h3,h4 arr	900	h1,h2,h3,h4		
cr = 0	1000			
h4 dep	1011,36	h1,h2,h3	111,36	-1084,38016
cr = 0	1250			
h3 dep	1261,36	h1,h2	361,36	-1084,38016
cr = 0	1500			
h2 dep	1511,36	h1	611,36	-1084,38016
L arr	1749			

cr = 0	1750			
h1,h2,h3,h4 arr	1800	h1,h2,h3,h4,h1		
L dep	1873,32			560,36608
h1 dep	1884,68	h1,h2,h3,h4	984,68	-524,01408
cr = 0	2000			
h4 dep	2011,36	h1,h2,h3	211,36	-1084,38016
cr = 0	2250			
h3 dep	2261,36	h1,h2	461,36	-1084,38016
cr = 0	2500			
h2 dep	2511,36	h1	711,36	-1084,38016
h1,h2,h3,h4 arr	2700	h1,h2,h3,h4,h1		
L arr	2749			
cr = 0	2750			
L dep	2873,32			560,36608
h1 dep	2884,68	h1,h2,h3,h4	1084,68	-524,01408

Figure 4.9: Calculation example, based on Sim 1.

The scheduling scenario for each cycle is the same as the worst-case scheduling scenario for stream H, as shown in Figure 4.2. If the worst-case scheduling scenario occurs, and stream H experiences the minimum send interval, in consecutive cycles, it will only take 2884,68 μ s before stream H exceeds the deadline. We can see that setting the send interval 100 μ s below the send interval (period) used to calculate the bandwidth reservation used, results in a 100 μ s increase in queue times per cycle, given that the worst-case scheduling scenario occurs each cycle. In the remainder of the example, shown in Appendix E, I also show that the queue times are constant when using a send interval of 1000 μ s, and that the queue times decrease by 100 μ s per cycle when setting the send interval to 1100 μ s. Figure 4.9 shows one out of several scheduling scenarios which causes a buildup in queue times, and an eventual breach of the deadline.

At this point I want to note a difference between the utilization constraint and the deadline constraint, when calculating the bandwidth reservation for streams H and M, using Algorithm 1 and Algorithm 2 from [8]. In Experiment 1, the utilization constraint is used to reserve the bandwidth at all frame sizes for stream H, and up to but not including a frame size of 642 B for stream M, since it produces the greater value at these frame sizes. The bandwidth reservations for the remaining frame sizes for stream M is calculated using the deadline

constraint, and the increase in bandwidth reservation from the deadline constraint equation is meant to help against interfering traffic so that stream M can still meet the deadline requirement at frame sizes 642 B and above: the increasing transmission times due to frame size increase would eventually have caused the last stream M frame in the worst-case scenario (Figure 4.4) to break the deadline otherwise.

Now, when a stream has its bandwidth reserved according to the utilization constraint, the following holds true at every frame size for Experiment 1: if we start at $t = 0$ and have 4 frames arrive in the queue at the same time, it will take $1000\ \mu\text{s}$ before all 4 frames are forwarded from the queue and the credit value is recovered back to zero. $1000\ \mu\text{s}$ is the period used to calculate the bandwidth reservation. From this we can see that if frames arrive with a send interval less than $1000\ \mu\text{s}$ there will occur a buildup in queue times. From this, and the example in Figure 4.9, I conclude:

- if a stream is using the bandwidth reservation calculated by the utilization constraint, and the stream experiences a period of send interval times lower than the send interval used when calculating the bandwidth reservation for the stream, there may occur an accumulation in queue times. This applies to stream H at all frame sizes, and stream M up to and including a frame size of 542 B.
- I can also conclude that the stream may break the deadline sooner if it experiences the worst-case scheduling scenario; the worst-case scenario may be what pushes it over the edge, whereas it may not have broken the deadline during the simulation duration had it not experienced the worst-case scheduling scenario during a time period of lower send intervals. However, if there is a chance that there may occur an accumulation in queue times, however small, the deadline for the stream will eventually be breached, given a long enough simulation time. Then it becomes a conversation about how often this deadline breach will occur, what are the consequences when it does occur, and is the extra bandwidth required to avoid the deadline breach occurring better spent elsewhere.

Back to the deadline constraint: the bandwidth reservations calculated using the deadline constraint, from frame size 642 B and above for stream M, deviates further and further from the utilization constraint. Meaning, at each frame size increase, the difference between the deadline constraint reservation value and utilization constraint reservation value becomes larger. The result is that even if all sources use a send interval of $900\ \mu\text{s}$, stream M will not

break the deadline at a frame size of 1342 B, as long as the bandwidth reservation values are calculated based on a period of 1000 μ s. Stream M can, however, break the deadline at frame sizes 842 B, 742 B, and 642 B, when using a send interval of 900 μ s, which means that when running Sim 1, using a send interval variation range of [900,1100] μ s, there is a chance that stream M may break the deadline at these frame sizes, but not above. The deadline breach at frame sizes 842 B and below, can be seen when using Eq. (10) from [8] to calculate the required bandwidth reservation needed for stream M in order to not breach the deadline. Using Eq. (10) with a period of 900 μ s you would get a higher bandwidth reservation than what is currently reserved for a deadline of 1000 μ s, at frame sizes 842 B and below.

4.7 Sim 4: Increasing the bandwidth reservation

This simulation experiment is the same as Sim 1, but Instead of using the calculated bandwidth reservations as shown in Tables 4.1 and 4.2, I am in this experiment increasing the bandwidth reservation based on the lower bound of the send interval variation range [900,1100] μ s. This is the bandwidth reservation necessary to ensure that the deadline is never breached, at any frame size, for stream H, and up to and including a frame size of 842 B for stream M, when running Sim 1. However, an important thing to note here is that when increasing the bandwidth reservation for stream H: due to the reduction in send interval value I also need to increase the bandwidth reservation for stream M. This is because of Eq. (11) and the expression α^+_H / α^-_H in the denominator of this equation; a reduction in send interval (period) causes an increase in the α^+_H value and correspondingly a reduction in the α^-_H value, ultimately this causes the denominator in Eq. (11) to become smaller, which gives a larger bandwidth reservation value. In summary, when increasing the bandwidth reservation for stream H, in order to make sure the stream does not exceed the deadline when using a send interval variation range of [900,1100] μ s, I also need to increase the bandwidth reservation for stream M, according to the analysis in [8].

The bandwidth reservation for stream H is easily adjusted by multiplying the previously calculated bandwidth reservation values, based on a period of 1000 μ s, by a factor of (1/0.9). The stream M bandwidth reservation has to be recalculated using Eq. (11) from [8] with the newly calculated α^+_H values.

At a frame size of 1342 B, the source set is no longer schedulable with the increased bandwidth reservation values, as the combined reservation would then be $58,630356,427713256 \text{ Mb/s} + 47.715555556 \text{ Mb/s} = 104,313062716 \text{ Mb/s}$, thereby exceeding the link speed of 100 Mb/s. If I want to ensure that neither stream H nor stream M exceeds the deadline, in the case of a send interval variation range of [900,1100], I must use the bandwidth reservation values shown in the table below, it also means that I cannot schedule the source set at a frame size of 1342 B.

Frame size	Maximum delay H [μs]	Maximum delay M [μs]	Reservation H	Reservation M	Average delay H [μs]	Average delay M [μs]
142 B	653,36	657,36	5,048888889 Mb/s	5,048888889 Mb/s	137,902	144,963
242 B	661,36	708,68	8,604444444 Mb/s	8,604444444 Mb/s	146,119	153,944
342 B	669,36	708,68	12,16 Mb/s	12,16 Mb/s	155,05	163,387
442 B	677,36	708,68	15,715555554 Mb/s	15,715555554 Mb/s	163,994	172,775
542 B	685,36	783,68	19,271111109 Mb/s	19,271111109 Mb/s	173,433	182,945
642 B	693,36	783,68	22,826666664 Mb/s	22,826666664 Mb/s	183,038	193,165
742 B	701,36	783,68	26,382222222 Mb/s	26,382222222 Mb/s	193,512	205,403
842 B	709,36	783,68	29,937777775 Mb/s	29,937777775 Mb/s	204,412	217,576
942 B	717,36	783,68	33,493333333 Mb/s	34,058707884 Mb/s	216,421	226,869
1042 B	725,36	791	37,048888885 Mb/s	39,239437768 Mb/s	228,566	232,253
1142 B	733,36	707,156	40,604444444 Mb/s	44,961514990 Mb/s	242,927	243,429
1242 B	777,6	792,92	44,159999996 Mb/s	51,360958572 Mb/s	260,404	248,539

Table 4.15: Increased bandwidth reservation values, together with maximum delay and average delay times when using the increased bandwidth reservation values.

4.7.1 Addressing the low-priority traffic

As mentioned previously, the combined utilization of streams H and M, assuming a 1000 μ s send interval, at a frame size of 1342 B, is 85,845056 Mb/s, this leaves 14,154944 Mb/s for stream L, which only uses 12,336 Mb/s (assuming a send interval of 1000 μ s). The maximum delay experienced by an L frame is in Sim 1 at a frame size of 1342 B, 1521,19 μ s, and an average queue time of 302,37 μ s. In the tables below I show how increasing the bandwidth reservation, using the bandwidth reservation values shown in Table 4.15 for streams H and M, affects stream L.

Frame size	Maximum delay L [μs]	Average queue time L [μs]
142 B	148	0,82
242 B	164	2,69
342 B	180	5,22
442 B	196	9,74
542 B	212	13,98
642 B	228	15,63
742 B	244	21,61
842 B	259	28,16
942 B	275	36,02
1042 B	292	45,97
1142 B	308	56,91
1242 B	733,6	105,81

Table 4.16: Sim 1 (default seed-set) stream L statistics.

Frame size	Maximum delay L [μs]	Average queue time L [μs]
142 B	148	0,6368
242 B	162	2,19
342 B	180	4,066
442 B	196	7,365
542 B	212	10,753
642 B	228	16,096
742 B	244	21,169
842 B	260	29,903
942 B	276	36,813
1042 B	291,317	48,498
1142 B	308	58,683
1242 B	910,92	121,048

Table 4.17: Sim 1 (default seed-set) stream L data. Using increased bandwidth reservation values for stream H and M, shown in Table 4.15.

An increase in bandwidth reservation means a larger idleSlope, and thereby a smaller sendSlope. This translates to the cost of forwarding a frame from the queues being less, and the credit generating back to zero faster. Meaning, L traffic has less time throughout the simulation where it is eligible for forwarding, as stream L frames can only be forwarded when the credit value for streams H and M are negative. Thereby, an increase in bandwidth reservation for streams H and M results in average longer wait times and maximum wait times for stream L frames.

4.8 Sim 5: constant low-priority traffic

Up until now, the L source has sent traffic with a send interval of 1000 μ s, using a \pm variation of 10, 5, and 2 %. In this simulation experiment there will always be low-priority traffic present in queue 4. This means that whenever the credit value for queue 5 (M) and queue 6 (H) is negative at the same time, an L frame will be forwarded. Apart from this modification in L traffic, this simulation experiment is the same as Sim 1, using a send interval variation range for streams H and M of [900,1100].

In order to record the effect of having constant low priority traffic in queue 4, I use a warm-up period. according to the OMNeT++ manual ([56]): “The warmup-period option specifies the length of the initial warm-up period. When set, results belonging to the first x seconds of the simulation will not be recorded into output vectors, and will not be counted into the calculation of output scalars.”

Instead of using only one L source, as in the previous simulation experiments, I here use two L sources in order to fill up queue 4 during the warm-up period.

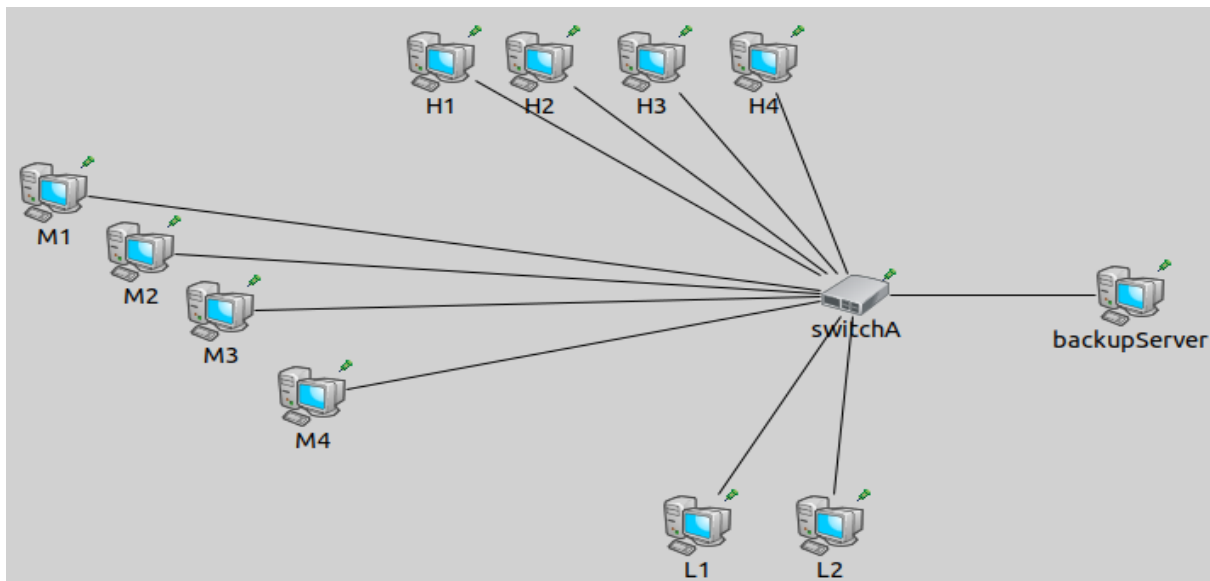


Figure 4.10: Sim 5 topology.

A problem arises due to having used, up until this point, the same source type, VlanEtherHostQ in my simulations. This source type consists of several submodules, amongst them the trafGenApp module which inherits from the VlanEtherTrafGen module which again extends the EtherTrafGen module. Now, the EtherTrafGen module is where I implement the code for generating pseudorandom start times and send interval times (see Modification 3 in section 3.4).

In order to have the stream H and M sources send with a variable send interval rate, and sources L1 and L2 send with a constant rate (to fill up queue 4), I have created a new source type, using VlanEtherHostQ as a template, named VlanEtherHostQ2, which I use as sources L1 and L2. This change means that there in this simulation experiment will only be 8 sources of type VlanEtherHostQ generating random start times and send-interval times, not 9 as in the previous simulation experiments. This skews the generated send interval times, compared to Sim 1 which I will compare the simulation data to:

Send interval cycle	H1	H2	H3	H4	L1	M1	M2	M3	M4
1	988 μ s	970 μ s	936 μ s	939 μ s	958 μ s	1093 μ s	1040 μ s	987 μ s	921 μ s
2	1065 μ s	981 μ s	1074 μ s	970 μ s	925 μ s	909 μ s	977 μ s	988 μ s	987 μ s

Table 4.18: Send-intervals generated for the first two cycles from Sim 1.

Send interval cycle	H1	H2	H3	H4	M1	M2	M3	M4
1	988 μ s	970 μ s	936 μ s	1093 μ s	958 μ s	1040 μ s	987 μ s	921 μ s
2	981 μ s	988 μ s	987 μ s	972 μ s	1065 μ s	925 μ s	1074 μ s	939 μ s

Table 4.19: Send-intervals generated for the first two cycles from Sim 5.

This makes it harder to compare this simulation experiment directly with Sim 1, in order to see exactly how much the constant low priority traffic impacts the queue times.

In this simulation experiment I have L1 and L2 start sending at $t = 0$, both sources send at an interval of 123,36 μ s, which is the time it takes to forward 1542 B over a 100 Mb/s link. This traffic pattern continues for both sources up until $t = 255 \mu$ s, at which point source L2 stops sending for the remainder of the simulation. At this point there are 3 low priority frames in queue 4, source L1 then continues to send frames with an interval of 123,36 μ s for the remainder of the simulation time. The warm-up period is set to 255 μ s. And as we can see from the queue length vector below, from the time I start recording statistics to the end of the simulation, queue 4 is never empty:

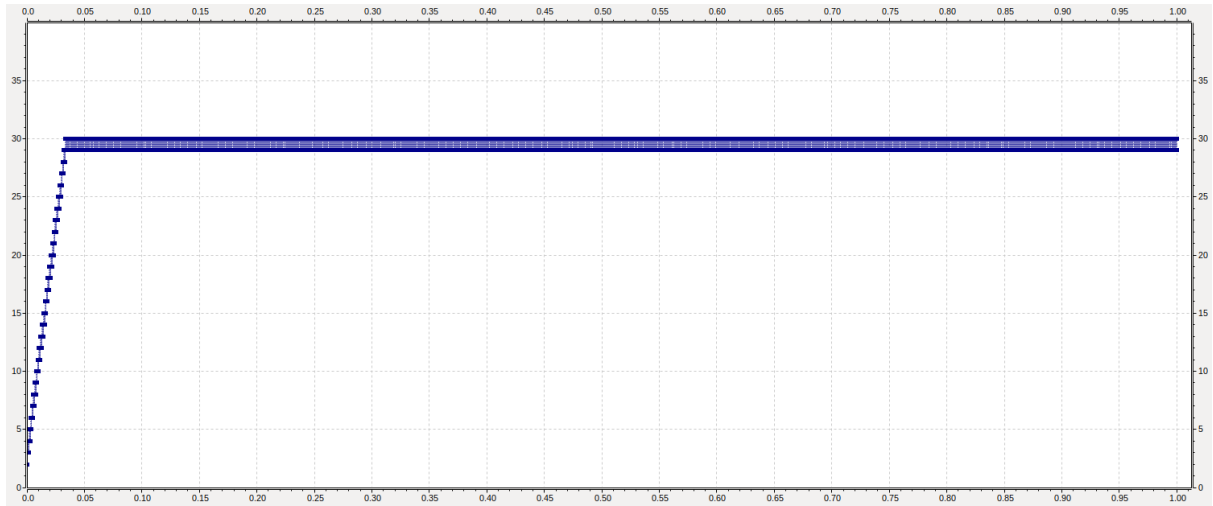


Figure 4.11: Number of frames in queue 4 during Sim 5.

With the configured buffer size (370080 b), the queue can only hold 30 frames of size 1542 B.

H1	H2	H3	H4	M1	M2	M3	M4
999,862 μs	1000,01 5 μs	999,6163 8361 μs	1001,257 51503 μs	1002,1533 0661 μs	998,7102 8971 μs	999,719 μs	999,8791 2087 μs

Table 4.20: Send interval averages for Sim 5, default seed-set, range [900,1100].

Total average send interval for stream H is 1000,18772466 μs , and for stream M is 1000,115429298 μs . The maximum delay and average delay times for streams H and M from running this simulation is shown below, I here use the original bandwidth reservation values, as shown in Tables 4.1 and 4.2.

Frame size	Maximum delay H [μs]	Maximum delay M [μs]	Average delay H [μs]	Average delay M [μs]
142 B	1407,04	1357,36	563,19	627,66
242 B	1408,12	1454,6	570,3	639,2
342 B	1385,08	1439,87	577,98	650,93
442 B	1411,84	1400,75	585,02	662,79
542 B	1438,24	1426,91	592,13	674,7
642 B	1421,56	1219,07	599,89	485,12
742 B	1413,15	884,92	606,87	293,02

842 B	1420,56	844,36	615,34	284,66
942 B	1422,23	830,92	623,25	284,56
1042 B	1481,63	828,52	632,09	286,48
1142 B	1436,0	852,52	640,87	288,87
1242 B	1499,08	772,44	649,72	293,68
1342 B	1450,91	912,28	658,25	314,16

Table 4.21: Maximum delay and average delay times for Sim 5, using default seed-set.

In order to more accurately quantify how the constant low priority traffic affects the delay times for stream H and M, I will run Sim1 again, using 8 VlanEtherHostQ sources (4 stream M sources and 4 stream H sources) with modification 3, same as in this simulation experiment, and 1 VlanEtherHostQ2 source sending low priority traffic with a constant send interval of 1000 μ s. By doing this I get the same send interval average for each source as in this simulation experiment, and the only difference is the stream L traffic. I will refer to this modified version of Sim 1 as “Sim 1 modified”. The data for running Sim 1 modified, is shown in the table below:

Frame size	Maximum delay H [μs]	Maximum delay M [μs]	Average delay H [μs]	Average delay M [μs]
142 B	1298,36	1310,36	508,05	569,2
242 B	1306,36	1318,36	520,53	576,66
342 B	1314,36	1326,36	531,33	584,15
442 B	1322,36	1334,36	541,55	591,67
542 B	1330,36	1342,36	551,83	599,27
642 B	1338,36	1074,01	552,52	420,12
742 B	1376,18	779,68	562,62	224,36
842 B	1380,1	710,0	573,19	217,32
942 B	1372,36	726,0	585,07	218,92
1042 B	1371,67	735,64	597,99	223,75
1142 B	1389,55	756,89	612,88	231,11

1242 B	1420,91	821,28	628,5	237,18
1342 B	1462,67	837,28	657,24	344,54

Table 4.22: Maximum delay and average delay times for Sim 1 modified.

From Table 4.21 and Table 4.22: we can see that when there is always low-priority traffic present in queue 4, the average delay times for both stream H and M is slightly higher, most notably for stream M, and for both streams the difference is larger at the lower frame sizes. Also, when always having low-priority traffic in queue 4, the maximum delay for both stream H and M, for most frame sizes, is around 100 μ s higher, the exceptions are 1342 B for stream H and 1242 B for stream M. Also, when it comes to difference in maximum delay, the difference is more prevalent for stream M, especially for some of the frame sizes with bandwidth reservations according to the deadline constraint equation, such as 642 B, 742 B and 842 B.

An event log detailing the scheduling scenario at the output port of the switch for stream H, when the first stream H frame at a frame size of 142 B breaks the deadline, for Sim 5, is shown in Appendix C (C.6). This event log can be compared with C.1, which details the first deadline breach for stream H from Sim 1, in order to get a better understanding about how constant L traffic affects the H traffic at the output port of the switch.

The total utilization of streams H and M is not affected by having constant low priority traffic in queue 4, only the maximum delay and average delay times. I provide utilization statistics for Sim 5 and Sim 1 modified to show this in Appendix D (D.3).

I also run Sim 5 with the increased bandwidth reservation values, shown in Table 4.15 in the previous section, confirming that neither stream exceed the deadline when using the increased bandwidth reservation values. Data for this is provided in Appendix D (D.4).

4.9 Preemption of low-priority traffic

Introduced in Chapter 2 of this thesis, frame preemption distinguishes between two types of queues: express and preemptible. Queues assigned as express may interrupt the transmission of queues assigned as preemptible, if preemption is enabled on the port. In a scenario where a frame in an express queue is ready to be forwarded, but there is currently a frame from a preemptible queue being forwarded by the switch, the express frame must wait a maximum of 123 octet times. On a 100 Mb/s link 123 octet times equals 9,84 μ s, but oftentimes the express frame will only have to wait 64 octet times or less [19]. A thing to consider when using frame preemption is that IEEE Std 802.3br ([35]) does not enforce a time limit on how long a preemptible frame can be preempted. Considering this, frame preemption seemingly fits well with CBS: assuming the SR classes are configured as express, and L traffic is configured as preemptible, the credit of the SR classes will become negative with regular intervals, thereby allowing low priority traffic to be forwarded.

A drawback by using preemption is that it causes the network to be utilized less efficiently; more of the traffic will be overhead as a result of splitting preemptible frames in one or several new frames. Another trade-off is that: while the express frames may reduce their delay by preempting preemptible frames, the delay of the preemptible frames are increased.

I will run Sim 5 (constant low priority traffic), but this time using frame preemption. I configure queues 5 (M) and 6 (H) as express, and queue 4 (L) as preemptive. Meaning, frames from streams H and M may preempt L frames. Data for this simulation experiment is provided in the table below:

Frame size	Maximum delay H [μs]	Maximum delay M [μs]	Average delay H [μs]	Average delay M [μs]
142 B	1298,36	1310,36	506,73	560,554
242 B	1306,36	1318,399	517,454	568,966
342 B	1314,36	1326,36	528,014	577,301
442 B	1322,36	1334,36	538,66	585,725

542 B	1330,679	1342,36	549,339	594,216
642 B	1338,36	1074,014	550,178	410,519
742 B	1377,468	780,96	560,088	215,736
842 B	1382,921	709,502	569,952	205,849
942 B	1373,647	720,96	580,084	204,787
1042 B	1376,459	687,92	590,907	205,489
1142 B	1390,838	663,452	601,568	206,924
1242 B	1404,96	677,96	612,869	209,36
1342 B	1420,96	690,56	630,501	221,382

Table 4.23: Data for Sim 5 with preemption: streams H and M may preempt L frames.

When compared to Sim 5 without preemption (Table 4.21), we can see that preemption lowers the maximum delay and average delay times for both streams, at all frame sizes. The maximum delay reduction ranges from 30 μ s to 222 μ s, and the average delay time reduction ranges from 27 to 88 μ s. The effect of preemption on the average delay times, and possibly maximum delay, will be greater in a network with a lot of low priority traffic to preempt. I provide a comparison for this, by running Sim 1, where the L source has a send interval of 1000 μ s \pm 10%, with preemption: streams H and M may preempt L traffic. Data for Sim 1 with preemption is provided in Appendix D (D.5).

In conclusion:

- frame preemption provides a reduction in average delay times, and may provide a reduction in maximum delay: the maximum delay in the simulation experiments, at some frame sizes, might not be due to L traffic, instead it might be solely due to an accumulation of frames in the queue due to send interval variation.

- As previously discussed, when there is send interval variation where the lower bound of the variation interval is less than what was used to reserve the bandwidth for the stream, frame preemption may increase the time before a deadline breach, and it may cause a breach to occur less frequent. But if there is a possibility of the deadline being breached, it will eventually be breached.
- While frame preemption may cause a reduction in maximum delay and average delay times for the express streams, it will also cause an increase in maximum delay and average delay times for the preemptive streams. Also, frame preemption will cause the network to be utilized less efficiently since more of the traffic going over the links will be overhead.

Chapter 5

Conclusion

This thesis has used the bandwidth reservation analysis from [8] to calculate bandwidth reservation values for AVB class A and B, in the presence of low priority traffic for a single Ethernet TSN switch, given the topology and parameters for Experiment 1, as described in [8]. It was then verified that the calculated bandwidth reservation values were enough to ensure that the deadline was not breached by the AVB streams, referred to as H and M. This verification was done by constructing the worst-case scheduling scenario, for each of the AVB streams, first through calculations, and then by implementing Experiment 1 into my chosen simulation model. The feasibility of the assumption made in [8], about the sources being synchronized to the rest of the network, with no variation in send interval, was then challenged. Through several simulation experiments, this thesis evaluated the effect of each source having a certain send interval variation range, given that the bandwidth reservation for each stream was calculated based on the median value of said send interval variation range. This thesis also looked at how constant low-priority traffic affected average and maximum delay times at the output port of the TSN switch for the AVB streams, and how the reserved bandwidth for the AVB streams affect queue times for the L traffic. In addition, this thesis briefly looked into frame preemption, and how this mechanism can be used by the AVB streams. Alongside the simulation experiments, some observations about the results have been pointed out and discussed.

This thesis has shown that when using the bandwidth reservation algorithm in [8], if the sources have variations in their send interval, and the bandwidth reservations are not based on the lower bound of the send interval variation range, there is a chance that the deadline will be breached, and therefore, given enough time, the deadline will be breached. As observed, this is true even if the send interval average for the stream is the same as, or slightly higher than, the send interval used when calculating the bandwidth reservation for the stream. Also, if the stream experiences the worst-case scheduling scenario at the output port of the TSN switch, this may cause the stream to break the deadline sooner than if it did not experience the worst-case scenario. However, the stream will eventually break the deadline regardless of the worst-case scenario happening or not.

Frame preemption may be used as a way of delaying the deadline breach from occurring, and reducing the probability of it occurring. However, using frame preemption causes the network to be utilized less efficiently due to more overhead being transmitted and more interframe spacing being required. In addition, while frame preemption lowers the maximum and average delay of streams using the express queue(s), it also increases the maximum and average delay of the preemptive queue(s).

If there is send interval variation, the bandwidth reservation of the stream has to be calculated based on the lower bound of the send interval variation range, in order to guarantee that the deadline is not breached. But having an absolute guarantee that the deadline will not be breached has consequences. There is already a large difference, at a lot of the frame sizes, between the average delay and the maximum delay. Depending on how often the deadline breach occurs, one has to establish the consequences when it does occur, depending on the application, and determine if the extra bandwidth required to avoid the deadline breach from occurring is better spent elsewhere. Also, when increasing the bandwidth reservation for the AVB streams, in order to secure the deadline, this affects the low-priority traffic by increasing average and maximum delay times, as shown in subsection 4.7.1. And, increasing the bandwidth reservation values may cause a reduction in the number of frame sizes the source set is schedulable at, as seen in section 4.7.

Chapter 6

Future work

This thesis had ambitions of expanding the bandwidth reservation analysis in [8] from single node to multi node, but due to time restrictions this did not occur. However, conclusions relevant for multi node can still be drawn from the single node analysis presented in this thesis, as the single node analysis shows the effect of send interval variations on the AVB class A and B queues, which inevitably will occur in a larger network. The next step was to construct a line topology of multiple switches, and recalculate the bandwidth reservation values, but distribute the deadline over the number of switches.

Bibliography

- [1] S. Vitturi, C. Zunino, and T. Sauter, "Industrial Communication Systems and Their Future Challenges: Next-Generation Ethernet, IIoT, and 5G," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 944-961, June 2019, doi: 10.1109/jproc.2019.2913443.
- [2] M. R. Palattella *et al.*, "Standardized Protocol Stack for the Internet of (Important) Things," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1389-1406, Third Quarter 2013, doi: 10.1109/surv.2012.111412.00158.
- [3] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17-27, March 2017, doi: 10.1109/MIE.2017.2649104.
- [4] H. Xu, W. Yu, D. Griffith, and N. Golmie, "A Survey on Industrial Internet of Things: A Cyber-Physical Systems Perspective," *IEEE Access*, vol. 6, pp. 78238-78259, 2018, doi: 10.1109/ACCESS.2018.2884906.
- [5] R. Schlesinger, A. Springer, and T. Sauter, "New approach for improvements and comparison of high performance real-time ethernet networks," *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, Spain, 2014*, pp. 1-4, doi: 10.1109/ETFA.2014.7005356.
- [6] P. Danielis, H. Puttnies, E. Schweissguth, and D. Timmermann, "Real- Time Capable Internet Technologies for Wired Communication in the Industrial IoT-a Survey," *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Turin, Italy, 2018*, pp. 266-273, doi: 10.1109/ETFA.2018.8502528.
- [7] P. Danielis *et al.*, "Survey on real-time communication via ethernet in industrial automation environments," *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, Spain, 2014*, pp. 1-8, doi: 10.1109/ETFA.2014.7005074.
- [8] J. Cao, M. Ashjaei, P. J. L. Cuijpers, R. J. Bril, and J. J. Lukkien, "An independent yet efficient analysis of bandwidth reservation for credit-based shaping," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, Imperia, Italy, 2018, pp. 1-10, doi: 10.1109/WFCS.2018.8402345.
- [9] N. Warden, "Overview and effect of deterministic ethernet on test strategies," *2017 IEEE AUTOTESTCON, Schaumburg, IL, USA, 2017*, pp. 1-3, doi: 10.1109/AUTEST.2017.8080468.
- [10] S. Kehrer, O. Kleineberg, and D. Heffernan, "A comparison of fault-tolerance concepts for IEEE 802.1 Time Sensitive Networks (TSN)," *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, Spain, 2014*, pp. 1-8, doi: 10.1109/ETFA.2014.7005200.
- [11] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks," *IET Cyber-Physical Systems: Theory & Applications*, vol. 1, no. 1, pp. 86-94, 2016, doi: 10.1049/iet-cps.2016.0021.

- [12] J. L. Messenger, "Time-Sensitive Networking: An Introduction," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 29-33, June 2018, doi: 10.1109/mcomstd.2018.1700047.
- [13] T. Garrett, L. E. Setenareski, L. M. Peres, L. C. E. Bona, and E. P. Duarte, "Monitoring Network Neutrality: A Survey on Traffic Differentiation Detection," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2486-2517, 2018, doi: 10.1109/COMST.2018.2812641.
- [14] M. D. Johas Teener *et al.*, "Heterogeneous Networks for Audio and Video: Using IEEE 802.1 Audio Video Bridging," *Proceedings of the IEEE*, vol. 101, no. 11, pp. 2339-2354, 2013, doi: 10.1109/JPROC.2013.2275160.
- [15] "IEEE Standard for Local and metropolitan area networks--Audio Video Bridging (AVB) Systems," *IEEE Std 802.1BA-2011*, pp. 1-45, 30 Sept. 2011, doi: 10.1109/IEEESTD.2011.6032690.
- [16] "IEEE Standard for Local and metropolitan area networks--Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)," *IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005)*, pp. 1-119, 30 Sept. 2010, doi: 10.1109/IEEESTD.2010.5594972.
- [17] G. M. Garner *et al.*, "IEEE 802.1 AVB and Its Application in Carrier-Grade Ethernet [Standards Topics]," *IEEE Communications Magazine*, vol. 45, no. 12, pp. 126-134, December 2007, doi: 10.1109/MCOM.2007.4395377.
- [18] H. Lee, J. Lee, C. Park, and S. Park, "Time-aware preemption to enhance the performance of Audio/Video Bridging (AVB) in IEEE 802.1 TSN," in *2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI)*, Wuhan, China, 2016, pp. 80-84, doi: 10.1109/CCI.2016.7778882.
- [19] "IEEE Standard for Local and Metropolitan Area Network--Bridges and Bridged Networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pp. 1-1993, 6 July 2018, doi: 10.1109/IEEESTD.2018.8403927.
- [20] "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks," *IEEE Std 802.1AS-2011*, pp. 1-292, 30 March 2011, doi: 10.1109/IEEESTD.2011.5741898.
- [21] "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams," *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, vol., no., pp. C1-72, 5 Jan. 2010, doi: 10.1109/IEEESTD.2009.5375704.
- [22] J. Migge, J. Villanueva, N. Navet, and M. Boyer, "Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks," *Proc. Embedded Real-Time Software and Systems (ERTSS 2018)*, January 2018.
- [23] M. Ashjaei, G. Patti, M. Behnam, T. Nolte, G. Alderisi, and L. Lo Bello, "Schedulability analysis of Ethernet Audio Video Bridging networks with scheduled traffic support," *Real-Time Systems*, vol. 53, no. 4, pp. 526-577, 2017, doi: 10.1007/s11241-017-9268-5.

- [24] G. Alderisi, G. Patti, and L. L. Bello, "Introducing support for scheduled traffic over IEEE audio video bridging networks," *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, Cagliari, Italy, 2013, pp. 1-9, doi: 10.1109/ETFA.2013.6647943.
- [25] L. Lo Bello and W. Steiner, "A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1094-1120, 2019, doi: 10.1109/JPROC.2019.2905334.
- [26] N. Finn, "Introduction to Time-Sensitive Networking," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 22-28, June 2018, doi: 10.1109/mcomstd.2018.1700076.
- [27] L. Zhao, F. He, E. Li, and J. Lu, "Comparison of Time Sensitive Networking (TSN) and TTEthernet," *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, London, UK, 2018, pp. 1-7, doi: 10.1109/DASC.2018.8569454.
- [28] E. Mohammadpour, E. Stai, and J.-Y. Le Boudec, "Improved Credit Bounds for the Credit-Based Shaper in Time-Sensitive Networking," *IEEE Networking Letters*, vol. 1, no. 3, pp. 136-139, 2019, doi: 10.1109/lnet.2019.2925176.
- [29] J. Imtiaz, J. Jasperneite, and L. Han, "A performance study of Ethernet Audio Video Bridging (AVB) for Industrial real-time communication," *2009 IEEE Conference on Emerging Technologies & Factory Automation*, Palma de Mallorca, Spain, 2009, pp. 1-8, doi: 10.1109/ETFA.2009.5347126.
- [30] V. Gavriluț, L. Zhao, M. L. Raagaard, and P. Pop, "AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN," *IEEE Access*, vol. 6, pp. 75229-75243, 2018, doi: 10.1109/ACCESS.2018.2883644.
- [31] "IEEE Standard for Local and Metropolitan Area Networks--Timing and Synchronization for Time-Sensitive Applications," *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, vol., no., pp. 1-421, 19 June 2020, doi: 10.1109/IEEESTD.2020.9121845.
- [32] "IEEE Standard for Local and metropolitan area networks -- Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, vol., no., pp. 1-57, 18 March 2016, doi: 10.1109/IEEESTD.2016.8613095.
- [33] J. Falk *et al.*, "NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++," *2019 International Conference on Networked Systems (NetSys)*, Munich, Germany, 2019, pp. 1-8, doi: 10.1109/NetSys.2019.8854500.
- [34] "IEEE Standard for Local and metropolitan area networks -- Bridges and Bridged Networks -- Amendment 26: Frame Preemption," *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)*, vol., no., vol. 1-52, 30 Aug. 2016, doi: 10.1109/IEEESTD.2016.7553415.
- [35] "IEEE Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic," *IEEE Std 802.3br-2016 (Amendment to IEEE Std 802.3-2015 as amended by IEEE Std 802.3bw-2015, IEEE Std 802.3by-2016, IEEE Std 802.3bq-2016, and IEEE Std 802.3bp-2016)*, vol., no., pp. 1-58, 14 Oct. 2016, doi: 10.1109/IEEESTD.2016.7592835.

- [36] "IEEE Standard for Local and metropolitan area networks--Frame Replication and Elimination for Reliability," *IEEE Std 802.1CB-2017*, vol., no., pp. 1-102, 27 Oct. 2017, doi: 10.1109/IEEESTD.2017.8091139.
- [37] "IEEE Standard for Local and metropolitan area networks--Bridges and Bridged Networks--Amendment 28: Per-Stream Filtering and Policing," *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)*, vol., no., pp. 1-65, 28 Sept. 2017, doi: 10.1109/IEEESTD.2017.8064221.
- [38] "IEEE Standard for Local and metropolitan area networks--Bridges and Bridged Networks--Amendment 29: Cyclic Queuing and Forwarding," *IEEE Std 802.1Qch-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd(TM)-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, IEEE Std 802.1Qbz-2016, and IEEE Std 802.1Qci-2017)*, vol., no., pp. 1-30, 28 June 2017, doi: 10.1109/IEEESTD.2017.7961303.
- [39] A. Nasrallah et al., "Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88-145, Firstquarter 2019, doi: 10.1109/COMST.2018.2869350.
- [40] I. Álvarez Vadillo, A. Ballesteros, M. Barranco, D. Gessner, S. Djerasevic, and J. Proenza, "Fault Tolerance in Highly Reliable Ethernet-Based Industrial Systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 977-1010, June 2019, doi: 10.1109/JPROC.2019.2914589.
- [41] I. Álvarez, L. Moutinho, P. Pedreiras, D. Bujosa, J. Proenza, and L. Almeida, "Comparing Admission Control Architectures for Real-Time Ethernet," *IEEE Access*, vol. 8, pp. 105521-105534, 2020, doi: 10.1109/ACCESS.2020.2999817.
- [42] M. Ashjaei, M. Sjödin, and S. Mubeen, "A novel frame preemption model in TSN networks," *Journal of Systems Architecture*, vol. 116, 2021, doi: 10.1016/j.sysarc.2021.102037.
- [43] H. Wang, L. Zeng, M. Li, and C. Yang, "A Protocol Conversion Scheme Between WIA-PA Networks and Time-Sensitive Networks," *2019 Chinese Automation Congress (CAC), 2019*, pp. 213-218, doi: 10.1109/CAC48633.2019.8996753.
- [44] B. Fang, Q. Li, Z. Gong, and H. Xiong, "Simulative Assessments of Credit-Based Shaping and Asynchronous Traffic Shaping in Time-Sensitive Networking," presented at the 2020 12th International Conference on Advanced Infocomm Technology (ICAIT), 2020.
- [45] "ns-3 | a discrete event simulator for Internet systems. <https://www.nsnam.org/>." (accessed April 2021).
- [46] "OMNeT++ Discrete Event Simulator. <https://omnetpp.org/>." (accessed April 2021).
- [47] J. Jiang, Y. Li, S. H. Hong, A. Xu, and K. Wang, "A Time-sensitive Networking (TSN) Simulation Model Based on OMNET++," *2018 IEEE International Conference on Mechatronics and Automation (ICMA), 2018*, pp. 643-648, doi: 10.1109/ICMA.2018.8484302.
- [48] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, "Work in Progress: Investigating the Effects of High Priority Traffic on the Best Effort Traffic in TSN Networks," presented at the 2019 IEEE Real-Time Systems Symposium (RTSS), 2019.

- [49] Z. Zhou, J. Lee, M. S. Berger, S. Park, and Y. Yan, "Simulating TSN traffic scheduling and shaping for future automotive Ethernet," *Journal of Communications and Networks*, vol. 23, no. 1, pp. 53-62, Feb. 2021, doi: 10.23919/JCN.2021.000001.
- [50] C. Le and D. Qiao, "Evaluation of Real-Time Ethernet with Time Synchronization and Time-Aware Shaper Using OMNeT++," *2019 IEEE 2nd International Conference on Electronics Technology (ICET)*, 2019, pp. 70-73, doi: 10.1109/ELTECH.2019.8839517.
- [51] T. Steinbach, H. D. Kenfack, F. Korf, and T. C. Schmidt, "An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy," *Proc. 4th Int. ICST Conf. Simulation Tools and Techniques (SIMUTools)*, pp. 375-382, 2011.
- [52] P. Meyer, T. Steinbach, F. Korf, and T. C. Schmidt, "Extending IEEE 802.1 AVB with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic," *2013 IEEE Vehicular Networking Conference, 2013*, pp. 47-54, doi: 10.1109/VNC.2013.6737589.
- [53] P. Heise, F. Geyer, and R. Obermaisser, "TSimNet: An Industrial Time Sensitive Networking Simulation Framework Based on OMNeT++," *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2016, pp. 1-5, doi: 10.1109/NTMS.2016.7792488.
- [54] "NeSTiNg - Network Simulator for Time-Sensitive Networking (TSN). <https://gitlab.com/ipvs/nesting>." (accessed April 2021).
- [55] D. Hellmanns, J. Falk, A. Glavackij, R. Hummen, S. Kehrer, and F. Dürr, "On the Performance of Stream-based, Class-based Time-aware Shaping and Frame Preemption in TSN," *2020 IEEE International Conference on Industrial Technology (ICIT)*, pp. 298-303, 2020, doi: 10.1109/ICIT45562.2020.9067122.
- [56] "OMNeT++ Simulation Manual. <https://doc.omnetpp.org/omnetpp/manual/>." (accessed April 2021).
- [57] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pp. 1-10, 2008, doi: 10.1145/1416222.1416290.
- [58] "What Is INET Framework? <https://inet.omnetpp.org/Introduction.html>." (accessed).
- [59] "Result Analysis with Python. <https://docs.omnetpp.org/tutorials/pandas/>." (accessed November 2020).
- [60] D. Vasilev, "Python programming training with the robot Finch," *2020 XXIX International Scientific Conference Electronics (ET)*, pp. 1-4, 2020, doi: 10.1109/ET50336.2020.9238231.
- [61] J. Cao, P. J. L. Cuijpers, R. J. Bril, and J. J. Lukkien, "Tight worst-case response-time analysis for ethernet AVB using eligible intervals," *2016 IEEE World Conference on Factory Communication Systems (WFCS)*, 2016, pp. 1-8, doi: 10.1109/WFCS.2016.7496507.
- [62] J. Cao, P. J. L. Cuijpers, R. J. Bril, and J. J. Lukkien, "Independent yet Tight WCRT Analysis for Individual Priority Classes in Ethernet AVB," presented at the Proceedings of the 24th International Conference on Real-Time Networks and Systems - RTNS '16, 2016.

- [63] "IEEE Standard for Ethernet," *IEEE Std 802.3-2015 (Revision of IEEE Std 802.3-2012)*, vol., no., pp.1-4017, 4 March 2016, doi: 10.1109/IEEESTD.2016.7428776.
- [64] N. Kero, A. Puhm, T. Kernen, and A. Mroczkowski, "Performance and Reliability Aspects of Clock Synchronization Techniques for Industrial Automation," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1011-1026, 2019, doi: 10.1109/JPROC.2019.2915972.
- [65] "IEEE Standard for Local and metropolitan area networks--Bridges and Bridged Networks," *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)*, vol., no., pp. 1-1832, 19 Dec. 2014, doi: 10.1109/IEEESTD.2014.6991462.
- [66] U. D. Bordoloi, A. Aminifar, P. Eles, and Z. Peng, "Schedulability analysis of Ethernet AVB switches," *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, 2014*, pp. 1-10, doi: 10.1109/RTCSA.2014.6910530.

Appendix A

Underlying work

The underlying topic in [8] is the appropriate allocation of network bandwidth in the context of industrial systems; appropriate in the sense that allocating too much bandwidth may cause inefficiency, but too little bandwidth may lead to a system failure. In short, [8] provides an algorithm for determining the minimum bandwidth reservation for the credit based shaper traffic classes A and B in a single Ethernet TSN switch, given a deadline constraint. This minimum bandwidth reservation allows each frame in streams A and B to meet the deadline, even in the worst-case scenario. In [8], AVB class A is referred to as H, and AVB class B is referred to as M, where H is high-priority traffic, and M is medium-priority traffic.

[8] compares its presented bandwidth reservation analysis to “the state-of-the-art algorithm”, found in [23], through experiments containing traffic flows of priority H, M, and L (best-effort). Due to multiple reasons, listed later in this Appendix, [8] observes that its new analysis outperforms the analysis in [23] regarding the efficiency in bandwidth reservation, at least in the experiments conducted in [8].

I want to make it clear that the bandwidth reservation analysis in [8] only applies to the output port of a single TSN switch, for traffic classes H and M, while taking into account L traffic. The analysis imposes a deadline requirement; after a frame enters the output port of the switch, it is required that the frame has been transmitted across the output port within the arbitrary deadline D_i . More specifically: the time a frame spends in its assigned queue + the time it takes the switch to transmit this frame across the output port cannot exceed D_i .

[8] refers to the IEEE Std 802.1Q from 2014 ([65]), Clause 34.4, where it is suggested to reserve the bandwidth of a priority class based on the expected utilization of that priority class. The now updated version of this IEEE standard [19], from 2018, makes the same suggestion, from what I can see in its Clause 34.4. [8] Then refers to a recent bandwidth reservation analysis performed in [23] where it is shown that the suggestion from the IEEE Std 802.1Q may cause traffic to become unschedulable. [8] Defines being schedulable as a source being able to meet its deadline requirement even in the worst-case scenario. [8] Points at findings in [23], which shows that assigning a higher bandwidth than what is suggested by

the IEEE Std 802.1Q may still help in terms of traffic being schedulable. As in [8], an algorithm to determine bandwidth reservation is presented in [23], this algorithm finds “a minimum bandwidth at each network link, at which the system is still schedulable” [8].

The work done in [23] uses the schedulability analysis performed in [66], which uses Busy Period Analysis to calculate the worst-case response times (WCRTs) for Ethernet AVB classes A and B. [8] references [61] and [62], which notes that the busy period analysis is often pessimistic due to the idling nature of credit-based shaping. As stated in [61]: “we abandon the busy period analysis because it easily leads to over-estimations when used on idling scheduling strategies”. [61] defines the busy period as “the longest interval in which there exists pending load.” And further states: “it is exclusively based on execution time and does not take the amount of provided bandwidth into account. As a consequence, it is less suitable for the study of idling systems.”

Due to the lack of suitability, as pointed to above, when using busy period analysis in order to calculate the worst-case response times, [61] introduces an alternative: eligible intervals. Eligible intervals can be defined in such a way that provisioning is taken into account [61]. Using eligible intervals instead of busy periods results in “a tight bound on the worst-case response times of credit-based shaping, independent of the knowledge of the inter-priority traffic except assumptions enforced by Ethernet TSN” [8]. A “tight bound” is defined in [62] to mean “no better estimates can be obtained unless more is known of the interfering traffic”.

The authors of [8], are the same authors behind [61] and [62], with the addition of one extra author in [8]. In [61] the authors proposed the alternative of eligible intervals to the use of busy periods, and show that by using eligible intervals tight worst-case response time bounds can more easily be obtained for Ethernet AVB, compared to using busy periods. However, this analysis is performed for either lower priority interference or higher priority interference, not both. In [62] the authors continue their work and provide an analysis for mixed interference. [8] uses the improved WCRT analysis in [62] to “determine a minimum bandwidth reservation for the credit-based shapers in a single switch.” This results in two algorithms, used to determine the bandwidth reservation for streams of class H and M, shown below this paragraph. Also, “two constraints are found in determining a minimum bandwidth reservation, i.e., deadline constraint and utilization constraint, with which it shows clearly when the utilization bound suggested by the standard is sufficient for schedulability and when a higher allocation is necessary.”

Algorithm 1 find α_H^+ for N_H sources in stream H

```

1: derive  $\alpha_H^+$  from Eq. (7)
2: for  $i = 1$  to  $N_H$  do
3:   derive  $newdc$  from Eq. (8)
4:    $\alpha_H^+ = \max(\alpha_H^+, newdc)$ 
5: end for
6: if  $\alpha_H^+ > BW$  then
7:   return FALSE
8: else
9:   return  $\alpha_H^+$ 
10: end if

```

Figure A.1: Bandwidth reservation algorithm for stream H. Obtained from [8].

$$\alpha_H^+ \geq U_H \cdot BW$$

Eq. (7), as shown in [8].

$$U_X = \sum_{\tau_i \in X} \frac{C_i}{T_i}$$

Utilization equation, as shown in [8].

$$\alpha_H^+ \geq \frac{\sum_{\tau_j \in H, j \neq i} C_j \cdot BW}{D_i - C_i - C_{M,L}^{max}}$$

Eq. (8), as shown in [8].

Algorithm 2 α_M^+ for N_M messages in stream M

```

1: derive  $\alpha_M^+$  from Eq. (10)
2: for  $i = 1$  to  $N_M$  do
3:   derive  $newdc$  from Eq. (11)
4:    $\alpha_M^+ = \max(\alpha_M^+, newdc)$ 
5: end for
6: if  $\alpha_M^+ > BW - \alpha_H^+$  then
7:   return FALSE
8: else
9:   return  $\alpha_M^+$ 
10: end if

```

Figure A.2: Bandwidth reservation algorithm for stream M. Obtained from [8].

$$\alpha_M^+ \geq U_M \cdot BW$$

Eq. (10), as shown in [8].

$$\alpha_M^+ \geq \frac{\sum_{\tau_j \in M, j \neq i} C_j \cdot BW}{D_i - C_i - C_L^{max} \cdot \left(1 + \frac{\alpha_H^+}{\alpha_H^-}\right) - C_H^{max}}$$

Eq. (11), as shown in [8].

$$\alpha_X^- = BW - \alpha_X^+$$

Eq. (1), as shown in [8].

α^+_H is the bandwidth reservation for class H, and α^+_M is the bandwidth reservation for class M. Notice that in order to determine α^+_M I need to use Eq. (11), which again uses α^+_H , therefore I need to determine the bandwidth reservation for class H before I determine the bandwidth reservation for class M. In this regard, [8] provides some information: “As α^+_H grows larger, more interference (worst-worst case relative delay) is introduced to stream M, see Theorem 1. Therefore, more reservation is needed for stream M to deal with the longer delay due to interference. Conversely, the minimum reservation of stream H consistently leads to the minimum reservation of stream M, refer to Eq. (11), which is in line with the objective to allocate the minimum bandwidth for streams H and M. Knowing the minimum value for α^+_H is therefore crucial to the last step to determine the schedulability of stream M, since the calculated minimum value of α^+_H cannot exceed the remaining bandwidth excluding stream H.”

[8] constructs 4 theoretical experiments, in order to illustrate how to apply the new algorithm, calculating the reservation values for stream H and M given the parameters introduced in each of the experiments. [8] also calculates the bandwidth reservation values for each experiment by using the algorithm in [23]. The calculated bandwidth reservation values, for each experiment, from using the algorithm in [23], are plotted together with the bandwidth reservation values calculated using its own algorithm, for comparison. In the 2 first experiments in [8] the utilization is varied gradually: by increasing the payload in Experiment 1 and by increasing the period in Experiment 2. In Experiment 3 they look at an arbitrary deadline vs. constrained deadline, and in Experiment 4 they look at non-identical sources.

Appendix B

Experiment 1 calculations

This Appendix details the bandwidth reservation calculations performed for Experimenting 1 from [8]. Note that this Appendix uses Algorithm 1 and Algorithm 2, and the equations used by these algorithms, from [8]. The algorithms and equations required for these calculations are also shown in the previous Appendix, A.

Below is a table of abbreviations and descriptions relevant when calculating the bandwidth reservation values for Experiment 1.

Abbreviation	Description
α^+_H	Bandwidth reservation value for class H.
α^+_M	Bandwidth reservation value for class M.
U_X	Utilization for class X.
T_i	Period of a source i.
C_i	Maximum transmission time for source i.
τ_i	Periodic source i.
C^{\max}_X	Maximum transmission time of a stream of class X.
D_i	Deadline for a source i.
BW	Bandwidth given by the link speed.
$C^{\max}_{M,L}$	$\max(C^{\max}_M, C^{\max}_L)$

Table B.1: Table of abbreviations with accompanying descriptions. Made from abbreviations and descriptions found in [8].

The general information for each experiment, as found in [8]: “In this subsection, we assume n identical periodic sources in stream H and n identical ones in stream M. Each source has a period T , a maximum transmission time C , and a deadline D .” Also, “in all experiments we assume a total bandwidth of 100 Mbps and set the maximum payload for stream L according to the Ethernet standard, 1500 Bytes, and thus the maximum transmission C_L is 123,36 μ s. For stream M, we use the known maximum transmission time of the stream H instead of the one from the standard.” Note that if the payload for the stream L frames are 1500 Bytes and the transmission time is 123,36 μ s over a 100 Mb/s link, this means that the size of the remaining frame must be 42 B:

$$(1542*8) \text{ b} / 100 \text{ Mb/s} = 12336 \text{ b} / 100\,000\,000 \text{ b/s} = 0,00012336 \text{ s} = 123,36 \mu\text{s}.$$

The specific information for Experiment 1 is: “In this experiment, the number of sources n is fixed to 4 for streams H and M, and each source has a fixed period and deadline $T = D = 1000\mu$ s. We vary the payload from 100 to 1500 Bytes with a step-size of 100 Bytes (the maximum transmission time C varies from 11,36 μ s to 123,36 μ s). The utilizations of streams H and M gradually increase to nearly 50%.”

I will start by calculating the bandwidth reservation for the first payload, 100 B, for stream H, using Algorithm 1, shown in Figure 2.10. On the first line, Algorithm 1 uses Eq. (7) to find a value for α^+_H , and Eq. (7) uses the Utilization equation. Calculating the utilization for stream H: for each periodic sources τ_i in class H, I take the maximum transmission time, C_i , and divide it by the period, T_i , and then add up the values calculated for each source. For the first source in stream H: $C_1 / T_1 = 11,36 \mu\text{s} / 1000 \mu\text{s} = 0,01136$. This number will be the same for each source since the payload and period is the same for all 4 sources in stream H (and for stream M). U_H is therefore equal to $0,01136 + 0,01136 + 0,01136 + 0,01136 = 0,01136 * 4 = 0,04544$. Moving back to Eq. (7): $\alpha^+_H \geq U_H * \text{BW} = 0,04544 * 100 \text{ Mb/s} = 4,544 \text{ Mb/s}$

I have now performed line 1 of Algorithm 1.

On line 2-5 Algorithm 1 uses a for-loop, where for each source in stream H the loop derives newdc from Eq. (8), the loop then sets α^+_H equal to the maximum value of the comparison between the α^+_H value found from Eq. (7), and the newdc value derived from Eq. (8) for source 1 in stream H. In the next iteration of the for-loop, newdc is derived from Eq. (8) for source 2 in stream H and compared with the previously found maximum value of the comparison between α^+_H found from Eq. (7), and newdc from Eq. (8) for source 1 in stream

H. This calculation and comparison is performed for each source in stream H, returning the maximum value of α^+_H .

When looking at Eq. (8) I notice that there is no “newdc”, as referred to in Algorithm 1, and 2. There is no mention of newdc elsewhere in [8] nor is it mentioned in any of the references. However, [8] does refer to Eq. (8) and Eq. (11) as the deadline constraint for stream H and M, respectively. I therefore interpret “derive newdc” in Algorithm 1 as finding the minimum value for α^+_H from Eq. (8) and “derive newdc” in Algorithm 2 as finding the minimum value for α^+_M from Eq. (11). Since the newdc value from Eq. (8) is the same for each source in stream H, for this experiment, the for-loop only has to be run once.

In the numerator of Eq. (8) the calculation: maximum transmission time multiplied with link speed is performed for each source in stream H, other than the current source (in the for-loop). In the denominator the maximum transmission time for the current source in stream H and the maximum transmission time for either stream M or L, whichever has the greatest, is subtracted from the deadline constraint for the current source in stream H. As stated earlier, since the deadline and starting payload is the same for each source in stream H, I only have to derive α^+_H from Eq. (8) once, I will show this calculation for source $i = 1$ in the for-loop. The numerator is $C_2 * BW + C_3 * BW + C_4 * BW$. The Denominator is $1000 \mu s - C_1 - C_L$. The maximum transmission time for stream L will always be greater than the maximum transmission time for stream M, with the exception of the final step size, where the payload of each sources in stream M will be 1500 B, which is the same value as the payload in stream L (held constant throughout Experiment 1). $C_L = (1542 * 8) b / 100 Mb/s = 0,00012336 s$.

Performing the calculation for source $i = 1$, using Eq. (8):

$$\begin{aligned}\alpha^+_H &\geq (C_2 * BW + C_3 * BW + C_4 * BW) / (1000 \mu s - C_1 - C_L) \\ \alpha^+_H &\geq (0,00001136 s * 100Mb/s + 0,00001136 s * 100 Mb/s + 0,00001136 s * 100 Mb/s) / \\ &(0,001 s - 0,00001136 s - 0,00012336 s) \\ \alpha^+_H &\geq 3408 b / 0,00086528 s = 3,938609467 Mb/s\end{aligned}$$

On line 6 of Algorithm 1 there is an if statement to check if the value found for α^+_H is greater than the link speed. As it does not make sense that stream (class) H can reserve more bandwidth than the link speed. If α^+_H is not greater than the link speed then Algorithm 1 returns the α^+_H value found in either Eq. (7) or Eq. (8), depending on which equation returns the highest α^+_H value. For Experiment 1, at a payload of 100 B (total frame size of 142 B), we can see that Eq. (7) produces the highest minimum bandwidth reservation value, $\alpha^+_H \geq 4,544 Mb/s$.

The bandwidth reservation values for stream H for Experiment 1 are shown in the graph below. Note that [8] does not list the exact bandwidth reservation values.

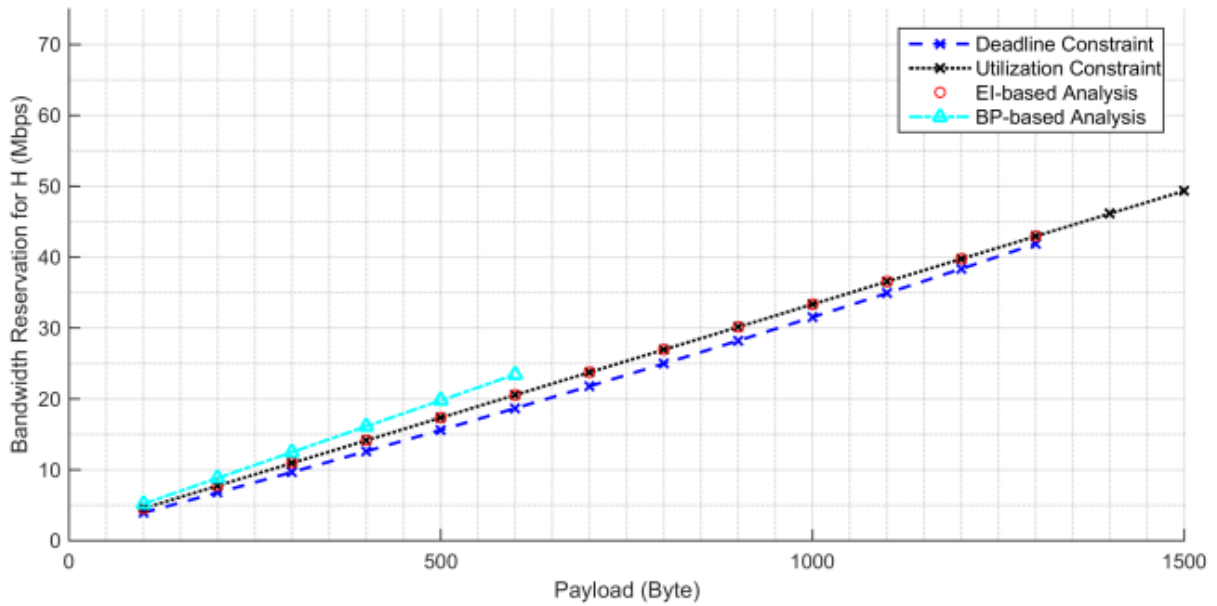


Figure B.1:Bandwidth reservation values for stream H. Obtained from [8].

The results from the new analysis in [8] is labeled with “EI-based Analysis”, while the results from the algorithm in [23], which [8] compares its experiment results against, is labeled with “BP-based Analysis”.

As I have now shown how the bandwidth reservation calculations for α^+_H is performed, according to Algorithm 1, for the first payload size of 100 B, I will simply list the α^+_H values calculated for the remaining payload sizes, as the process of calculation is the same at each step size. Keep in mind that the bandwidth reservation values are chosen from either Eq. (7) or Eq. (8), whichever produces the higher value, according to Algorithm 1. Note that the source set is not schedulable at payload sizes 1400 B and 1500 B, this is why I have only listed payload sizes up to and including 1300 B.

Payload	Eq. (7) (utilization constraint)	Eq. (8) (deadline constraint)
100 B	$\alpha^+_H \geq 4,544 \text{ Mb/s}$	$\alpha^+_H \geq \sim 3,938 \text{ Mb/s}$
200 B	$\alpha^+_H \geq 7,744 \text{ Mb/s}$	$\alpha^+_H \geq \sim 6,774 \text{ Mb/s}$
300 B	$\alpha^+_H \geq 10,944 \text{ Mb/s}$	$\alpha^+_H \geq \sim 9,664 \text{ Mb/s}$
400 B	$\alpha^+_H \geq 14,144 \text{ Mb/s}$	$\alpha^+_H \geq \sim 12,609 \text{ Mb/s}$

500 B	$\alpha^+_H \geq 17,344 \text{ Mb/s}$	$\alpha^+_H \geq \sim 15,610 \text{ Mb/s}$
600 B	$\alpha^+_H \geq 20,544 \text{ Mb/s}$	$\alpha^+_H \geq \sim 18,670 \text{ Mb/s}$
700 B	$\alpha^+_H \geq 23,744 \text{ Mb/s}$	$\alpha^+_H \geq \sim 21,789 \text{ Mb/s}$
800 B	$\alpha^+_H \geq 26,944 \text{ Mb/s}$	$\alpha^+_H \geq \sim 24,970 \text{ Mb/s}$
900 B	$\alpha^+_H \geq 30,144 \text{ Mb/s}$	$\alpha^+_H \geq \sim 28,214 \text{ Mb/s}$
1000 B	$\alpha^+_H \geq 33,344 \text{ Mb/s}$	$\alpha^+_H \geq \sim 31,524 \text{ Mb/s}$
1100 B	$\alpha^+_H \geq 36,544 \text{ Mb/s}$	$\alpha^+_H \geq \sim 34,902 \text{ Mb/s}$
1200 B	$\alpha^+_H \geq 39,744 \text{ Mb/s}$	$\alpha^+_H \geq \sim 38,349 \text{ Mb/s}$
1300 B	$\alpha^+_H \geq 42,944 \text{ Mb/s}$	$\alpha^+_H \geq \sim 41,867 \text{ Mb/s}$

Table B.2: Bandwidth reservation values for stream H.

Having calculated the bandwidth reservation values for stream H, I will now calculate the bandwidth reservation values for stream M. Algorithm 2 is very similar to Algorithm 1, but on the first line it derives α^+_M from Eq. (10), and in the for-loop it derives newdc from Eq. (11). It also performs a different if-test on line 6: instead of checking if the reserved bandwidth for class H is greater than the link speed, as in Algorithm 1, it checks if the combined reserved bandwidth for class H and M is greater than the link speed. This bandwidth check is not enforced by the standard, but [8] mentions it as a reasonable design choice and uses this assumption in their analysis.

Eq. (10) is the same as Eq. (7), but for stream M instead of stream H, since the payload and period is the same for both streams it follows that Eq. (10) will produce the same bandwidth reservation values as calculated for stream H from Eq. (7).

Deriving newdc from Eq. (11), as per line 3 in Algorithm 1, for the starting payload of 100 B: the numerator is the same as in Eq. (8), but the denominator is different, as this equation takes into account the influence of higher priority interference. α^-_H is found from Eq. (1):

$$\alpha^-_H = BW - \alpha^+_H = 100\text{Mb/s} - 4,544\text{Mb/s} = 95,456 \text{ Mb/s}$$

As the numerator in Eq. (11) is the same as the numerator in Eq. (8), and since the parameters are the same for both streams, I can use the same values as found for the numerator in Eq. (8) for stream H. Looking at the denominator of Eq. (11):

$$D_1 = 1000 \mu\text{s} = 0,001 \text{ s}. C_1 = 0,00001136 \text{ s}, C_L^{\max} = 1542 \text{ B} / 100 \text{ Mb/s} = 0,00012336 \text{ s}$$

$$1 + \alpha^+_H / \alpha^-_H = 1 + 4544000 \text{ b/s} / 95456000 \text{ b/s} = 1,047603084$$

I am assuming C_H^{MAX} is calculated for the current payload of 100 B, and is then equal to 0,00001136 s, same as C_1 .

$$\alpha^+_M \geq (C_2 * BW + C_3 * BW + C_4 * BW) / (D_1 - C_1 - C_L^{\text{max}} * (1 + \alpha^+_H / \alpha^-_H) - C_H^{\text{MAX}})$$

$$\alpha^+_M \geq 3408 \text{ b} / (0,001 \text{ s} - 0,00001136 \text{ s} - 0,00012336 \text{ s} * 1,047603084 - 0,00001136 \text{ s})$$

$$\alpha^+_M \geq 3408 \text{ b} / 0,000848048 = 4018640,454313907 \text{ b/s}$$

Since the α^+_M value found from Eq. (10) is greater than the α^+_M value found from Eq. (11), at the first frame size, the value from Eq. (10) is used, according to Algorithm 2.

Payload	Eq. (10) (utilization constraint)	Eq. (11) (deadline constraint)
100 B	$\alpha^+_M \geq 4,544 \text{ Mb/s}$	$\alpha^+_M \geq \sim 4,018 \text{ Mb/s}$
200 B	$\alpha^+_M \geq 7,744 \text{ Mb/s}$	$\alpha^+_M \geq \sim 7,018 \text{ Mb/s}$
300 B	$\alpha^+_M \geq 10,944 \text{ Mb/s}$	$\alpha^+_M \geq \sim 10,174 \text{ Mb/s}$
400 B	$\alpha^+_M \geq 14,144 \text{ Mb/s}$	$\alpha^+_M \geq \sim 13,503 \text{ Mb/s}$
500 B	$\alpha^+_M \geq 17,344 \text{ Mb/s}$	$\alpha^+_M \geq \sim 17,025 \text{ Mb/s}$
600 B	$\alpha^+_M \geq 20,544 \text{ Mb/s}$	$\alpha^+_M \geq 20,597966672682428 \text{ Mb/s}$
700 B	$\alpha^+_M \geq 23,744 \text{ Mb/s}$	$\alpha^+_M \geq 24,750211602634574 \text{ Mb/s}$
800 B	$\alpha^+_M \geq 26,944 \text{ Mb/s}$	$\alpha^+_M \geq 29,01684751939554 \text{ Mb/s}$
900 B	$\alpha^+_M \geq 30,144 \text{ Mb/s}$	$\alpha^+_M \geq 33,608448493209333 \text{ Mb/s}$
1000 B	$\alpha^+_M \geq 33,344 \text{ Mb/s}$	$\alpha^+_M \geq 38,580089785717592 \text{ Mb/s}$
1100 B	$\alpha^+_M \geq 36,544 \text{ Mb/s}$	$\alpha^+_M \geq 44,002196256730853 \text{ Mb/s}$
1200 B	$\alpha^+_M \geq 39,744 \text{ Mb/s}$	$\alpha^+_M \geq 49,967060764089695 \text{ Mb/s}$
1300 B	$\alpha^+_M \geq 42,944 \text{ Mb/s}$	$\alpha^+_M \geq 56,597507165186769 \text{ Mb/s}$

Table B.3: Bandwidth reservation values for stream M.

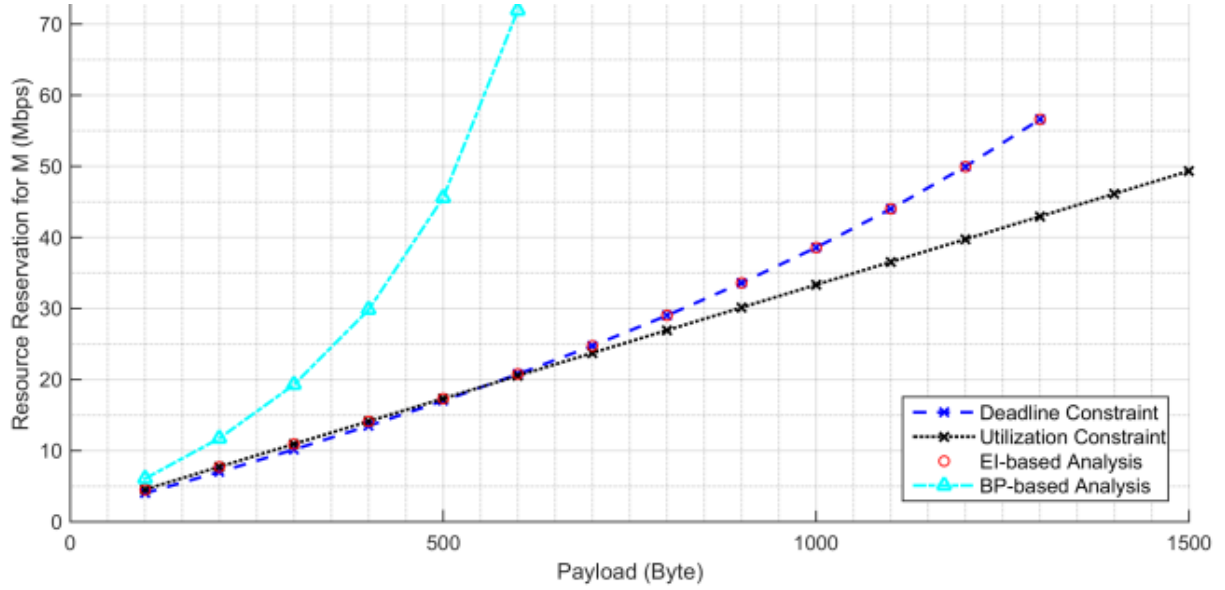


Figure B.2: Bandwidth reservation values for stream M. Obtained from [8].

At a payload of 600 B Eq. (11) starts giving a higher α_M^+ value than Eq. (10), as seen Figure 2.13 above. According to [8]: “For stream H, the deadline constraint gets closer to, but does not exceed the utilization when the payload increases. For stream M, the deadline constraint is initially smaller than and surpasses the utilization when the payload continues to grow. Our algorithm alternates its choice of utilization constraint and deadline constraint.”

The reason there are 13 red dots (EI-based Analysis) in Figure 2.12 and Figure 2.13 is because at a payload of 1400 B, α_H^+ would have been greater than or equal to 46,144Mb/s and α_M^+ would have been greater than or equal to $\sim 64,062\text{Mb/s}$. $64,062\text{Mb/s} + 46,144\text{Mb/s} > 100\text{Mb/s}$.

Appendix C

Event logs for scheduling scenarios

Note that while the “dep” event in section 4.2 marked when a frame was finished transmitting across the output port of the switch, below the “forwarded” event below marks when a frame starts being forwarded across the output port of the switch. As to avoid any confusion.

C.1 Sim 1: $\pm 10\%$ send interval variation (default seed-set)

Stream M first deadline breach event log (142 B frame size)

The most recent frame forwards were queue 6 at $t = 247 \text{ ms } 901,46 \mu\text{s}$ and queue 5 at $t = 247 \text{ ms } 932,46 \mu\text{s}$. For both frame forwards the credit value went from 0 to -1084.

- $t = 248 \text{ ms } 20 \mu\text{s}$: frame **M2** arrives in queue 5, there are now 4 frames in queue 5: first in line is frame M3, then M4, then M1, then M2.
- $t = 248 \text{ ms } 151 \mu\text{s}$: frame H4 is forwarded from queue 6, it has spent $566 \mu\text{s}$ in queue 6. There are now 2 frames in queue 6: first in line is H3, then H1. The credit value for queue 6 goes from 0 to -1084.
- $t = 248 \text{ ms } 182 \mu\text{s}$: frame M3 is forwarded from queue 5, it has spent $550 \mu\text{s}$ in queue 5. There are now 3 frames in queue 5: first in line is M4, then M1, then M2. The credit value for queue 5 goes from 0 to -1084.
- $t = 248 \text{ ms } 401 \mu\text{s}$: frame H3 is forwarded from queue 6, it has spent $733 \mu\text{s}$ in queue 6. There is now 1 frame in queue 6: H1. The credit value for queue 6 goes from 0 to -1084.
- $t = 248 \text{ ms } 432 \mu\text{s}$: frame M4 is forwarded from queue 5, it has spent $541 \mu\text{s}$ in queue 5. There are now 2 frames in queue 5: first in line is M1, then M2. The credit value for queue 5 goes from 0 to -1084.
- $t = 248 \text{ ms } 651 \mu\text{s}$: frame H1 is forwarded from queue 6, it has spent $639 \mu\text{s}$ in queue 6. There are now 2 frames in queue 6: first in line is H2, then H4. The credit value for queue 6 goes from 0 to -1084.
- $t = 248 \text{ ms } 682 \mu\text{s}$: frame M1 is forwarded from queue 5, it has spent $704 \mu\text{s}$ in queue 5. There are now 2 frames in queue 5: first in line is M2, then M3. The credit value for queue 5 goes from 0 to -1084.

- $t = 248 \text{ ms } 901 \text{ } \mu\text{s}$: frame H2 is forwarded from queue 6, it has spent $327 \text{ } \mu\text{s}$ in queue 6. There are now 2 frames in queue 6: first in line is H4, then H3. The credit value for queue 6 goes from 0 to -1084.
- $t = 248 \text{ ms } 922 \text{ } \mu\text{s}$: frame L1 is forwarded from queue 4.
- $t = 249 \text{ ms } 46 \text{ } \mu\text{s}$: frame **M2** is forwarded from queue 5, it has spent $1026 \text{ } \mu\text{s}$ in queue 5, this frame is the first frame in stream M to exceed the deadline. Adding the transmit time of $11,36 \text{ } \mu\text{s}$, the maximum delay for this frame is $1037,36 \text{ } \mu\text{s}$.

Send interval mean for each source in stream M leading up to the first deadline breach :

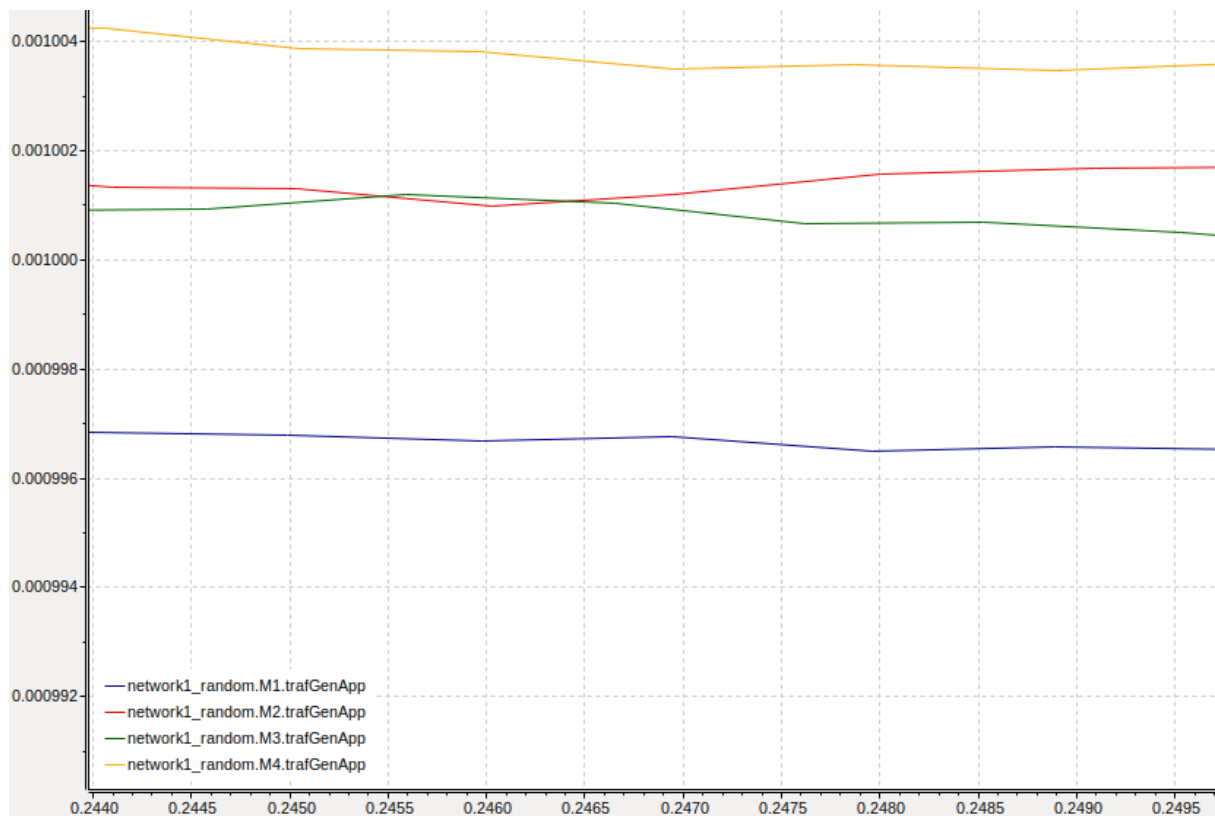


Figure C.1: Send interval mean for stream M, leading up to and including time of first deadline breach for stream M, at frame size = 142 B. Axes are in seconds.

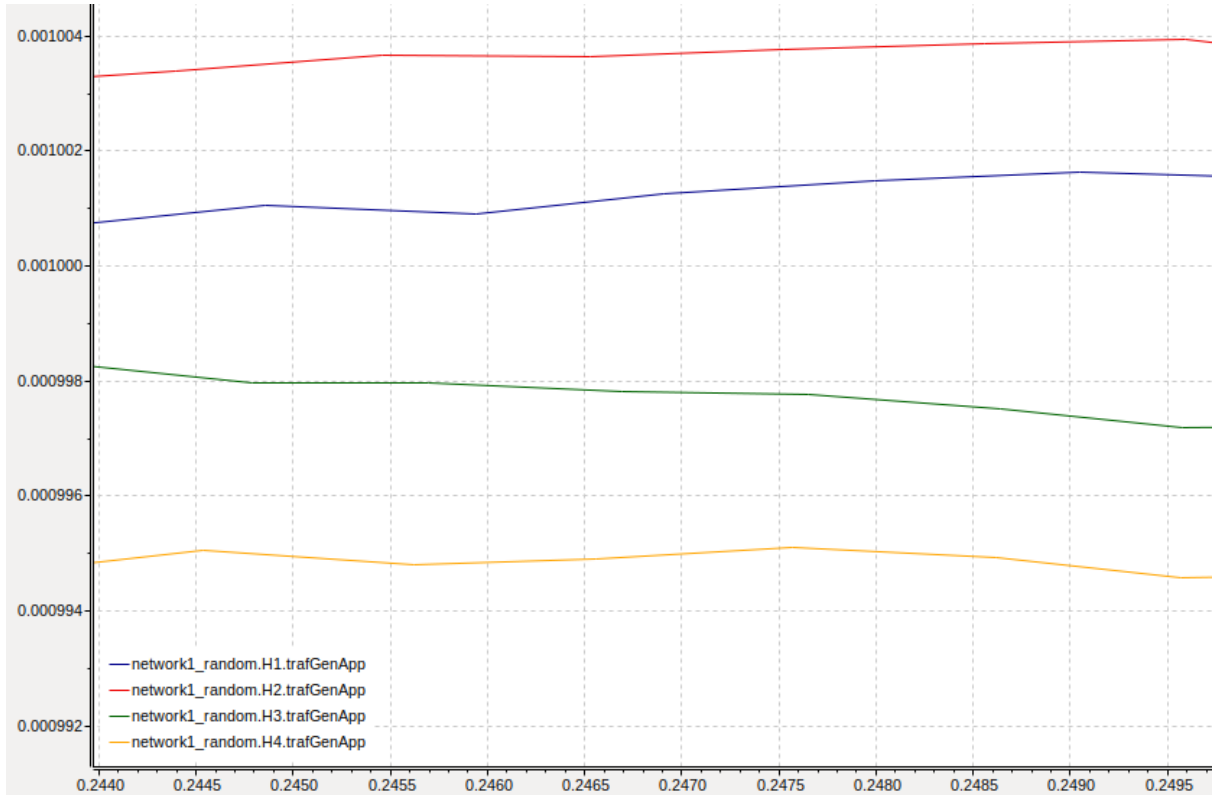


Figure C.2: Send interval mean for stream H, leading up to and including time of first deadline breach for stream M, at frame size = 142 B. Axes are in seconds.

The largest delay for stream H during the time period shown in Figure C.2 was 811,36 μ s.

Stream H first deadline breach event log (142 B frame size)

The previous frame forward from queue 6 occurred at $t = 265$ ms 401 μ s, and from queue 5 at $t = 265$ ms 182 μ s. For both frame forwards the credit value went from 0 to -1084.

- $t = 265$ ms 410 μ s: frame H1 arrives in queue 6, there are now 4 frames in queue 6: first in line is H2, then H3, then H4, then H1.
- $t = 265$ ms 432 μ s: frame M1 is forwarded from queue 5, it has spent 449 μ s in queue 5. There are now 3 frames in queue 5: first in line is M4, then M2, then M3. The credit value for queue 5 goes from 0 to -1084.
- $t = 265$ ms 651 μ s: frame H2 is forwarded from queue 6, it has spent 489 μ s in queue 6. There are now 3 frames in queue 6: first in line is H3, then H4, then H1. The credit value for queue 6 goes from 0 to -1084.
- $t = 265$ ms 682 μ s: frame M4 is forwarded from queue 5, it has spent 489 μ s in queue 5. There are now 2 frames in queue 5: first in line is M2, then M3. The credit value for queue 6 goes from 0 to -1084.
- $t = 265$ ms 894 μ s: frame L1 is forwarded from queue 4.

- $t = 266 \text{ ms } 18 \text{ } \mu\text{s}$: frame H3 is forwarded from queue 6, it has spent $759 \text{ } \mu\text{s}$ in queue 6. There are now 2 frames in queue 6: first in line is H4, then H1. Note that since the credit value for queue 6 reached 0 while frame L was transmitting, the credit value continued to generate above zero while waiting for interfering traffic. This caused the credit value for queue 6 to reach a value of 533 at the time of forwarding this frame. Since the cost of forwarding a frame at this reservation/frame-size is 1084, the credit value for queue 6 goes to -551 after forwarding this frame.
- $t = 266 \text{ ms } 31 \text{ } \mu\text{s}$: frame M2 is forwarded from queue 5, it has spent $661 \text{ } \mu\text{s}$ in queue 5. There are now 2 frames in queue 5: first in line is M3, then M1. Note that since the credit value for queue 5 reached zero and then had to wait for interfering traffic (first frame L and then frame H3), at the time of forwarding this frame the credit value for queue 5 is 448. Since The credit value for queue 5 goes to -636 after forwarding this frame.
- $t = 266 \text{ ms } 151 \text{ } \mu\text{s}$: frame H4 is forwarded from queue 6, it has spent $759 \text{ } \mu\text{s}$ in queue 6. The credit value for queue 6 goes from 0 to -1084. There is now 1 frame in queue 6: H1. This is the frame that will experience the greatest queue time and be the first frame in stream H to break the deadline, which we will see shortly.
- $t = 266 \text{ ms } 182 \text{ } \mu\text{s}$: frame M3 is forwarded from queue 5, it has spent $773 \text{ } \mu\text{s}$ in queue 5. There is now 1 frame in queue 5: M1. The credit value for queue 5 goes from 0 to -1084.
- $t = 266 \text{ ms } 401 \text{ } \mu\text{s}$: frame **H1** is forwarded from queue 6, it has spent $991 \text{ } \mu\text{s}$ in queue 6. The credit value for queue 6 goes from 0 to -1084. Now, the deadline is $1000 \text{ } \mu\text{s}$, which includes the time it takes to forward this frame across the output port of the switch, this time is $11,36 \text{ } \mu\text{s}$ at a frame size of 142 B, making the total $1002,36 \text{ } \mu\text{s}$ and thereby breaking the deadline.

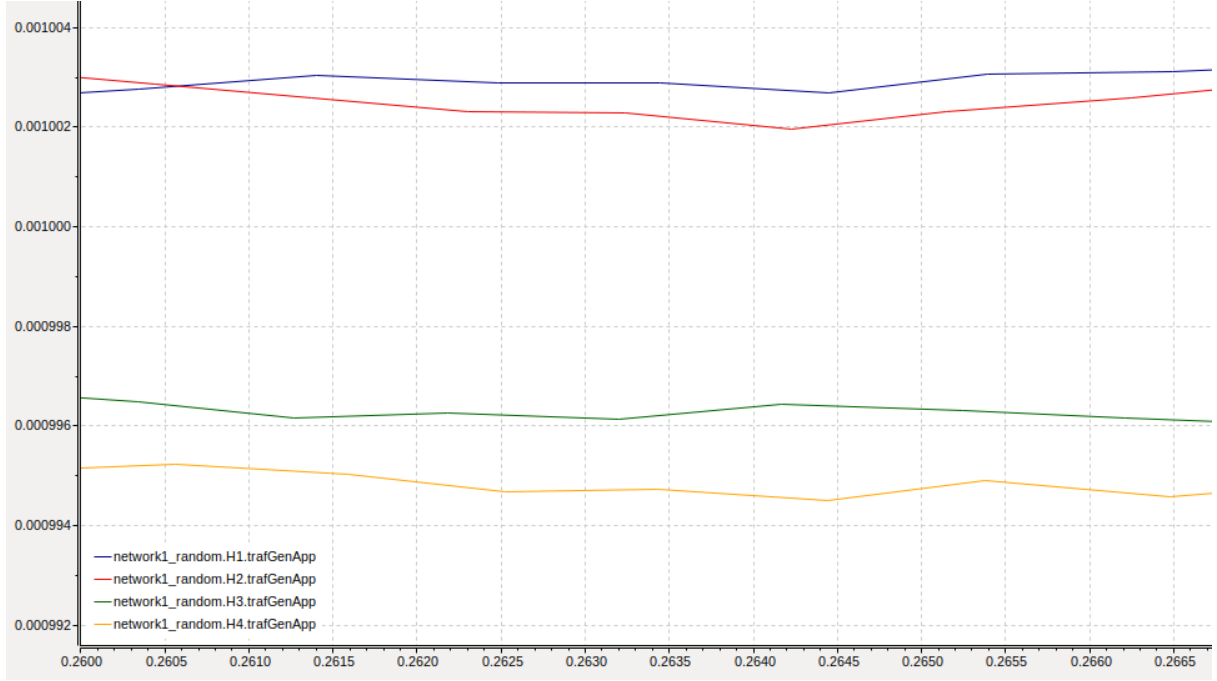


Figure C.3: Send interval mean for stream H, leading up to and including time of first deadline breach for stream H, at frame size = 142 B. Axes are in seconds.

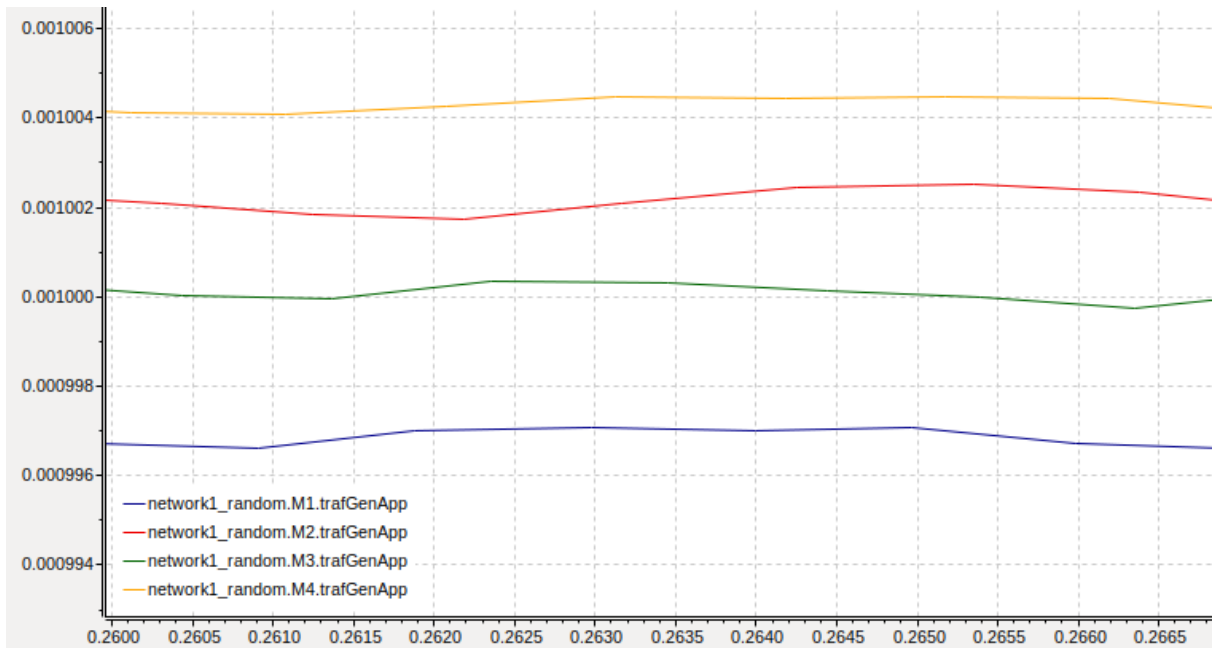


Figure C.4: Send interval mean for stream M, leading up to and including time of first deadline breach for stream H, at frame size = 142 B. Axes are in seconds.

The highest delay for a stream M frame during the time period shown in Figure C.4 was 826,36 μ s.

C.2 Sim 2: $\pm 5\%$ send interval variation (default seed-set)

Stream H first deadline breach event log (142 B frame size)

The last frame forwarded from queue 5 was at $t = 463 \text{ ms } 418 \text{ }\mu\text{s}$, and from queue 6 at $t = 463 \text{ ms } 222 \text{ }\mu\text{s}$. For both frame forwards the credit value went from 0 to -1084.

- $t = 463 \text{ ms } 451 \text{ }\mu\text{s}$: a frame from source **H1** arrives in queue 6. There are now 5 frames in queue 6: first in line is H4, then H2, then H3, then another frame from H4, then H1.
- $t = 463 \text{ ms } 472 \text{ }\mu\text{s}$: frame H4 is forwarded from queue 6. The credit value goes from 0 to -1084. There are now 4 frames in queue 6: first in line is H2, then H3, then H4, then H1.
- $t = 463 \text{ ms } 668 \text{ }\mu\text{s}$: frame M2 is forwarded from queue 5. The credit value goes from 0 to -1084. There are now 2 frames in queue 5: first in line is M3, then M1.
- $t = 463 \text{ ms } 722 \text{ }\mu\text{s}$: frame H2 is forwarded from queue 6. The credit value goes from 0 to -1084. There are now 3 frames in queue 6: first in line is H3, then H4, then H1.
- $t = 463 \text{ ms } 918 \text{ }\mu\text{s}$: frame M3 is forwarded from queue 5. The credit value goes from 0 to -1084. There are now 2 frames in queue 5: first in line is M1, then M4.
- $t = 463 \text{ ms } 972 \text{ }\mu\text{s}$: frame H3 is forwarded from queue 6. The credit value goes from 0 to -1084. There are now 2 frames in queue 6: first in line is H4, then H1.
- $t = 464 \text{ ms } 67 \text{ }\mu\text{s}$: frame L1 is forwarded from queue 4.
- $t = 464 \text{ ms } 191 \mu\text{s}$: frame M1 is forwarded from queue 5. The credit value goes from 106 to -978 (credit value went positive since it had to wait for interfering traffic). There are now 2 frames in queue 5: first in line is M4, then M2.
- $t = 464 \text{ ms } 222 \text{ }\mu\text{s}$: frame H4 is forwarded from queue 6. The credit value goes from 0 to -1084. There are now 2 frames in queue 6: first in line is H1, then H2.
- $t = 464 \text{ ms } 418 \text{ }\mu\text{s}$: frame M4 is forwarded from queue 5. The credit value goes from 0 to -1084. There are now 2 frames in queue 5: first in line is M2, then M3.
- $t = 464 \text{ ms } 472 \text{ }\mu\text{s}$: frame **H1** is forwarded from queue 6. It has spent $1021 \text{ }\mu\text{s}$ in queue 6 and is the first frame in stream H to breach the deadline. The credit value goes from 0 to -1084.

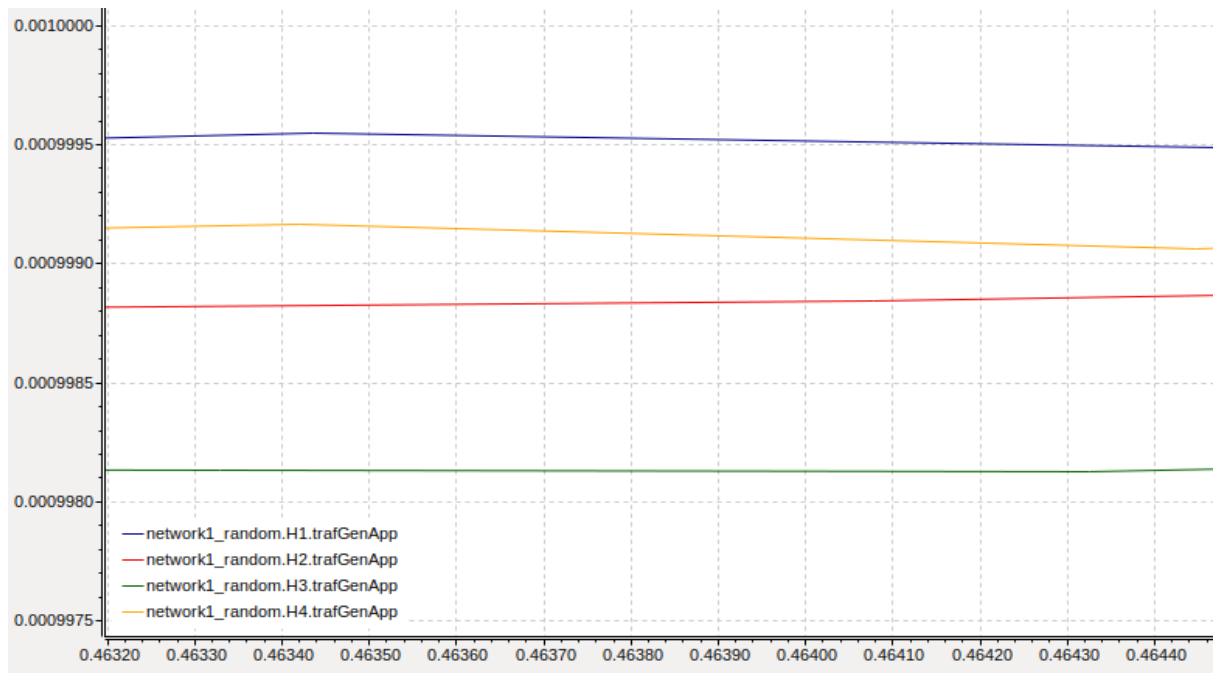


Figure C.5: Send interval mean for stream H, leading up to and including time of first deadline breach for stream H, at frame size = 142 B. Axes are in seconds.

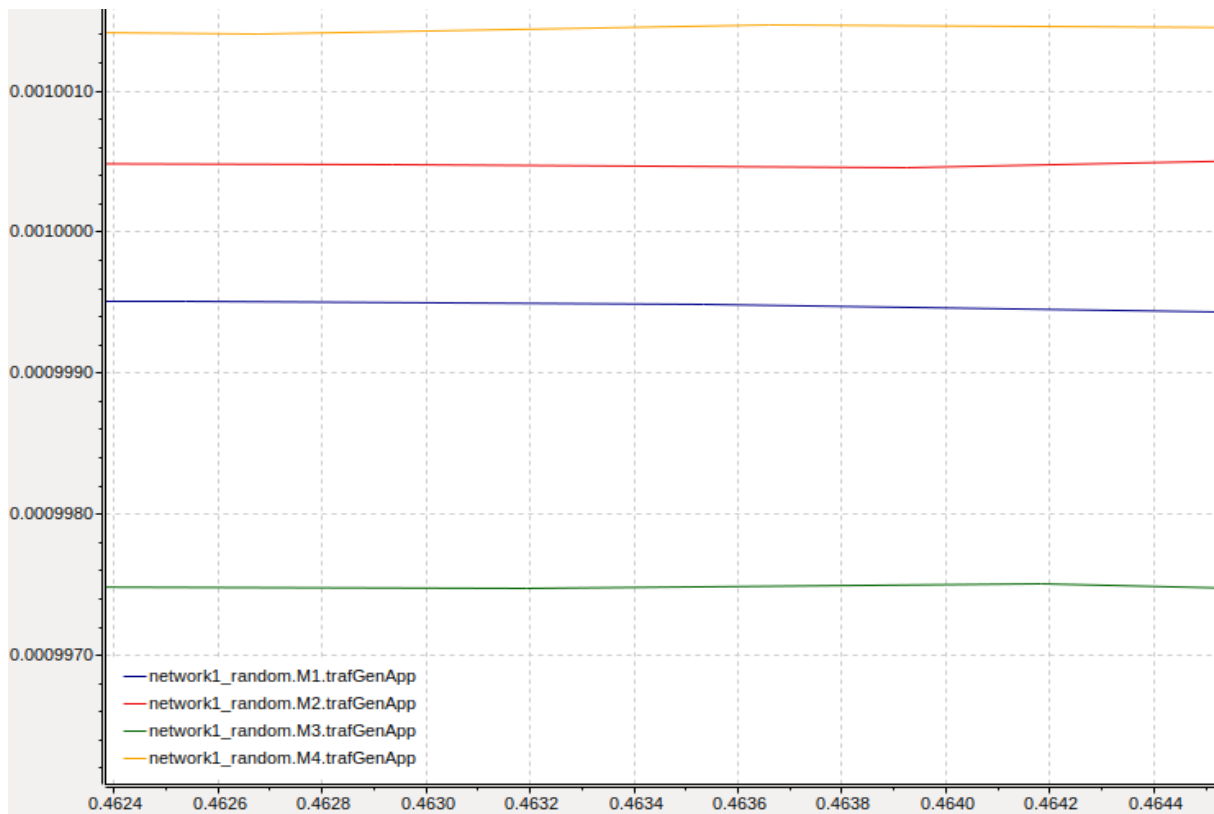


Figure C.6: Send interval mean for stream M, leading up to and including time of first deadline breach for stream H, at frame size = 142 B. Axes are in seconds.

Stream M, which does not break the deadline during the time period shown in Figures C.4 and C.5, experiences significantly higher send interval times.

Stream M greatest delay event log (142 B frame size)

- $t = 575 \text{ ms } 958 \text{ } \mu\text{s}$: a frame from source **M2** arrives in queue 5. There are now 4 frames in queue 5: first in line is M1, then M3, then M4, then M2.
- $t = 575 \text{ ms } 972 \text{ } \mu\text{s}$: frame H4 is forwarded from queue 6. The credit value goes from 0 to -1084. There are now 2 frames in queue 6: first in line is H2, then H1.
- $t = 576 \text{ ms } 168 \text{ } \mu\text{s}$: frame M1 is forwarded from queue 5. The credit value goes from 0 to -1084. There are now 3 frames in queue 5: first in line is M3, then M4, then M2.
- $t = 576 \text{ ms } 222 \text{ } \mu\text{s}$: frame H2 is forwarded from queue 6. The credit value goes from 0 to -1084. There are now 3 frames in queue 6: first in line is H1, then H4, then H3.
- $t = 576 \text{ ms } 362 \text{ } \mu\text{s}$: frame L1 is forwarded from queue 4.
- $t = 567 \text{ ms } 487 \text{ } \mu\text{s}$: frame H1 is forwarded from queue 6. The credit value goes from 65 to -1019, the credit accumulated above 0 when waiting for frame L1. There are now 3 frames in queue 6: first in line is H4, then H3, then H2.
- $t = 576 \text{ ms } 499 \text{ } \mu\text{s}$: frame M3 is forwarded from queue 5. The credit value goes from 367 to -717. There are now 2 frames in queue 5: first in line is M4, then M2.
- $t = 576 \text{ ms } 668 \text{ } \mu\text{s}$: frame M4 is forwarded from queue 5. The credit value goes from 0 to -1084. There are now 2 frames in queue 5: first in line is M2, then M1.
- $t = 576 \text{ ms } 722 \text{ } \mu\text{s}$: frame H4 is forwarded from queue 6. The credit goes from 0 to -1084. There are now 2 frames in queue 6: first in line is H3, then H2.
- $t = 576 \text{ ms } 918 \text{ } \mu\text{s}$: frame **M2** is forwarded from queue 5, it has spent $960 \text{ } \mu\text{s}$ in queue 5, adding the time it takes to transmit it from switchA I get: $960 \text{ } \mu\text{s} + 11,36 \text{ } \mu\text{s} = 971,36 \text{ } \mu\text{s}$. The credit value goes from 0 to -1084.

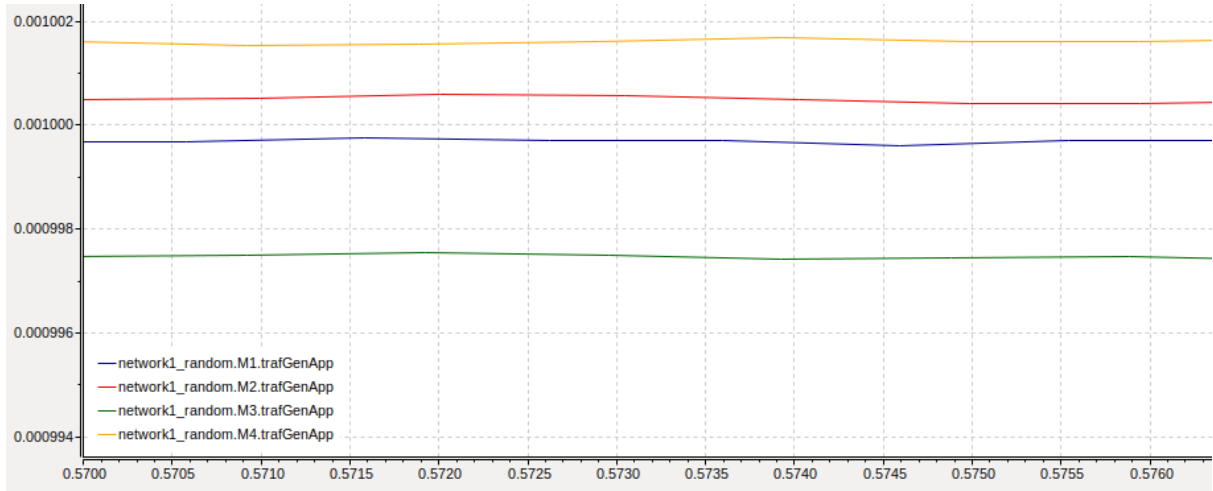


Figure C.7: Send interval mean for stream M, leading up to and including time of greatest delay experienced by stream M, at frame size = 142 B. Axes are in seconds.

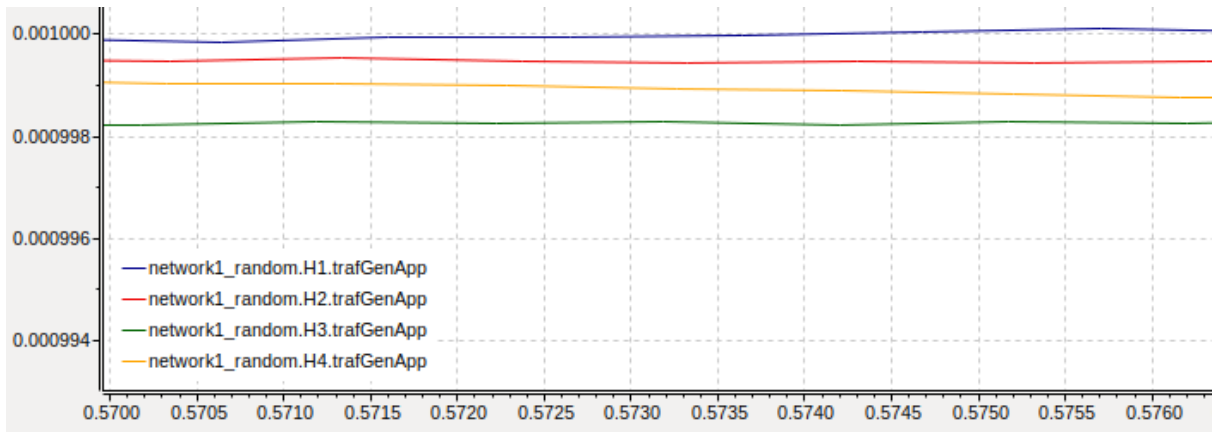


Figure C.8: Send interval mean for stream H, leading up to and including time of greatest delay experienced by stream M, at frame size = 142 B. Axes are in seconds.

During the time period shown in Figure C.7, stream H experiences a maximum delay of 919,36 μ s.

C.3 Sim 3: $\pm 2\%$ send interval variation (default seed-set)

Stream M greatest delay event log (142 B frame size)

In the average-case simulation experiment with a send interval variation of $1000 \mu\text{s} \pm 20 \mu\text{s}$ (2%) using default seed-set, stream M experiences a slightly larger maximum delay (537,36 μ s) than stream H (535,68 μ s). The maximum delay for stream M occurs at $t = 665 \text{ ms } 441 \mu\text{s}$ and for stream H at $t = 550 \text{ ms } 969 \mu\text{s}$.

- $t = 664 \text{ ms } 915 \mu\text{s}$: a frame from source **M4** arrives in queue 5. Queue 5 now contains 3 frames: first in line is M3, then M1, then M4.

- $t = 664 \text{ ms } 920 \text{ } \mu\text{s}$: frame L1 is forwarded from queue 4.
- $t = 665 \text{ ms } 4 \text{ } \mu\text{s}$: frame H1 is forwarded from queue 6. The credit value goes from 420 to -664. The credit value accumulated above 0 since the queue had to wait for the interfering frame L1. There are now 2 frames in queue 6: first in line is H2, then H4.
- $t = 665 \text{ ms } 57 \text{ } \mu\text{s}$: frame M3 is forwarded from queue 5. The credit value goes from 526 to -558. The credit value accumulated above 0 since the queue had to wait for the interfering frame L1 and H1. There are now 2 frames in queue 5: first in line is M1, then M4.
- $t = 665 \text{ ms } 191 \text{ } \mu\text{s}$: frame M1 is forwarded from queue 5. The credit value goes from 0 to -1084. There is now 1 frame in queue 5: M4.
- $t = 665 \text{ ms } 203 \text{ } \mu\text{s}$: frame H2 is forwarded from queue 6. The credit value goes from 6 to -1078. There is now 1 frame left in queue 6: H4.
- $t = 665 \text{ ms } 441 \text{ } \mu\text{s}$: frame **M4** is forwarded from queue 5, it has spent $526 \text{ } \mu\text{s}$ in the queue, if I add the time it takes to transmit the frame across the output port of switchA I get $526 \text{ } \mu\text{s} + 11,36 \text{ } \mu\text{s} = 537,36 \text{ } \mu\text{s}$. The credit value goes from 0 to -1084. The queue is now empty.

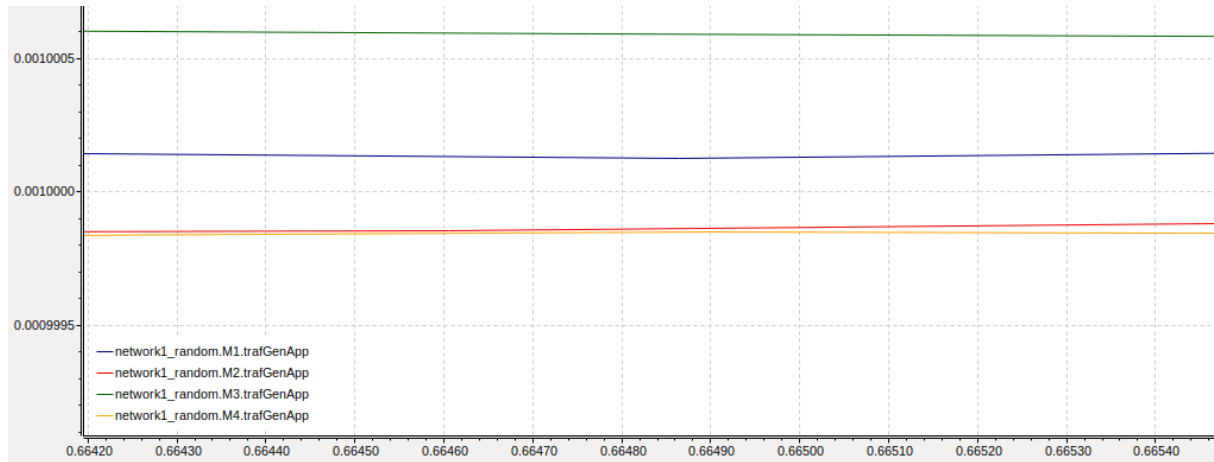


Figure C.9: Send interval mean for stream M, leading up to and including time of greatest delay experienced by stream M, at frame size = 142 B. Axes are in seconds.

C.4 Sim 3: $\pm 2\%$ send interval variation (seed-set 1)

Stream H greatest delay event log (142 B frame size)

In the average-case with 2% variation and seed-set 1, at a frame size of 142 B, stream H experiences the largest maximum delay, 835,36 μ s, at a simulation time of $t = 662 \text{ ms } 378 \text{ } \mu\text{s}$, below I will look at the scheduling scenario occurring for this frame.

- $t = 662 \text{ ms } 46 \text{ } \mu\text{s}$: a frame from source **H3** arrives in queue 6. There are now 4 frames in queue 6: first in line is H2, then H4, then H1, then H3. This is the frame experiencing the greatest maximum delay in stream H.
- $t = 662 \text{ ms } 86 \text{ } \mu\text{s}$: frame M4 is forwarded from queue 5. The credit value goes from 388 to -696. The credit value accumulates above 0 due to the interfering L frame. There is now 1 frame in queue 5: M3.
- $t = 662 \text{ ms } 120 \text{ } \mu\text{s}$: frame H2 is forwarded from queue 6. The credit value goes from 0 to -1084. There are now 3 frames in queue 6: first in line is H4, then H1, then H3.
- $t = 662 \text{ ms } 251 \text{ } \mu\text{s}$: frame M3 is forwarded from queue 5. The credit value goes from 0 to -1084. There are now 2 frames in queue 5: first in line is M2, then M1.
- $t = 662 \text{ ms } 370 \text{ } \mu\text{s}$: frame H4 is forwarded from queue 6. The credit value goes from 0 to -1084. There are now 2 frames in queue 6: first in line is H1, then H3.
- $t = 662 \text{ ms } 501 \text{ } \mu\text{s}$: frame M2 is forwarded from queue 5. The credit value goes from 0 to -1084. There are now 2 frames in queue 5: first in line is M1, then M4.
- $t = 662 \text{ ms } 620 \text{ } \mu\text{s}$: frame H1 is forwarded from queue 6. The credit value goes from 0 to -1084. There is now 1 frame in queue 6: H3.
- $t = 662 \text{ ms } 751 \text{ } \mu\text{s}$: frame M1 is forwarded from queue 5. The credit value goes from 0 to -1084. There is now 1 frame in queue 5: M4.
- $t = 662 \text{ ms } 870 \text{ } \mu\text{s}$: frame **H3** is forwarded from queue 6, it has spent 824 μ s in queue 5, adding the time it takes to transmit the frame across the output port of switchA I get $824 \text{ } \mu\text{s} + 11,36 \text{ } \mu\text{s} = 835,36 \text{ } \mu\text{s}$. The credit value goes from 0 to -1084. There is now 1 frame in queue 6: H2.

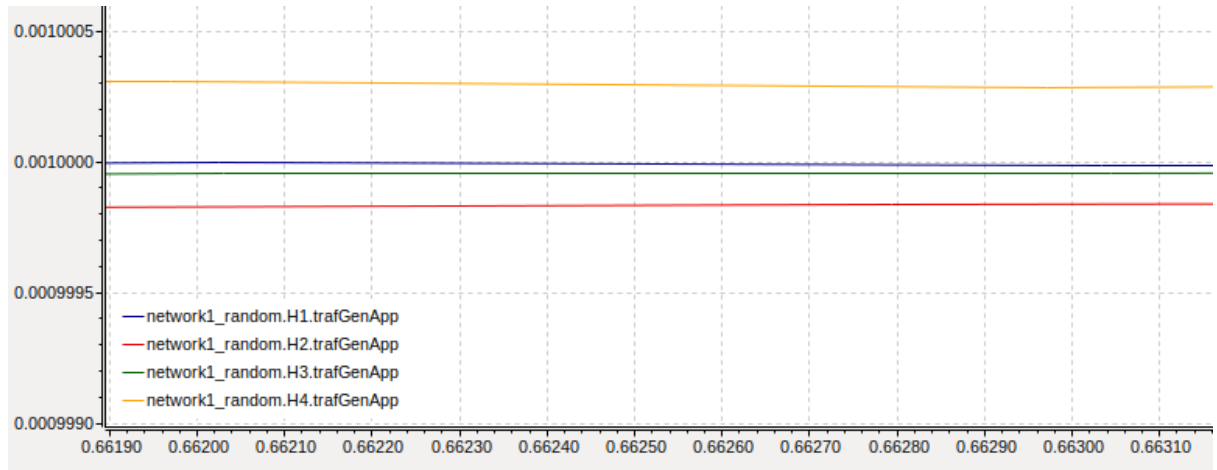


Figure C.10: Send interval mean for stream M, leading up to and including time of greatest delay experienced by stream M, at frame size = 142 B. Axes are in seconds.

C.5 Remarks about previously shown event logs and graphs

It seems like the most important thing for the delay a frame experiences are the number of frames in front of it in the queue (inter priority interference). In the logs above, if a frame has to wait for 4 frames ahead of it when arriving in the queue, it will breach the deadline, assuming the credit value is lower than 51,61984 when the frame arrives: at a frame size of 142 B, the cost of sending is 1084,38016, it takes 11,36 μ s to forward a frame, and 238,64 before the credit goes from -1084,38016 and back to zero. Meaning, if the credit starts at zero it will take $(11,36 + 238,64) \times 4 \mu\text{s} = 1000 \mu\text{s}$, to forward 4 frames, the fifth frame will then breach the deadline. The credit has to be at 51,61984 or above, to allow for 5 frame forwards without breaching the deadline. If there are interfering traffic, it may push the frame over the deadline earlier. If the credit is negative when the frame arrives in the queue, and it encounters interfering traffic right before the queue has enough credit to forward the frame, it may breach the deadline with only 3 frames in front of it in the queue when it arrives. The trend in the graphs is that lower send interval means results in lower queue times.

C.6 Sim 5: constant low-priority traffic

Stream H first deadline breach event log (142 B frame size)

Frame h1 arrives in queue 6 at $t = 331\text{ms } 511\mu\text{s}$. The previous frame transfer from queue 6 was at $t = 331\text{ms } 443\mu\text{s}$, the credit value went from 37 to -1047, meaning that the credit value will reach 0 at $t = 331\text{ms } 684\mu\text{s}$

- $t = 331\text{ms } 511\mu\text{s}$: a frame from source **H1** arrives in queue 6. There are now 4 frames in queue 6: first in line is H2, then H3, then H4, then H1.
- $t = 331\text{ms } 592\mu\text{s}$: an L frame is forwarded from queue 4.
- $t = 331\text{ms } 716\mu\text{s}$: frame H2 is forwarded from queue 6. The credit value goes from 142 to -942. There are now 3 frames in queue 6: first in line is H3, then H4, then H1.
- $t = 331\text{ms } 729\mu\text{s}$: an M frame is forwarded from queue 5. The credit value goes from 485 to -599.
- $t = 331\text{ms } 741\mu\text{s}$: an L frame is forwarded from queue 4.
- $t = 331\text{ms } 865\mu\text{s}$: an L frame is forwarded from queue 4.
- $t = 331\text{ms } 990\mu\text{s}$: frame H3 is forwarded from queue 6. The credit value goes from 248 to -836. There are now 2 frames in queue 6: first in line is H4, then H1.
- $t = 332\text{ms } 2\mu\text{s}$: an M frame is forwarded from queue 5. The credit value goes from 590 to -494.
- $t = 332\text{ms } 14\mu\text{s}$: an L frame is forwarded from queue 4.
- $t = 332\text{ms } 139\mu\text{s}$: an M frame is forwarded from queue 5. The credit value goes from 75 to -1009.
- $t = 332\text{ms } 151\mu\text{s}$: an L frame is forwarded from queue 4.
- $t = 332\text{ms } 275\mu\text{s}$: frame H4 is forwarded from queue 6. The credit value goes from 410 to -674. There is now 1 frame in queue 6: H1.
- $t = 332\text{ms } 287\mu\text{s}$: an L frame is forwarded from queue 4.
- $t = 332\text{ms } 412\mu\text{s}$: an M frame is forwarded from queue 5. The credit value goes from 181 to -903.
- $t = 332\text{ms } 424\mu\text{s}$: an L frame is forwarded from queue 4.
- $t = 332\text{ms } 548\mu\text{s}$: frame **H1** is forwarded from queue 6, it has spent $1037,48\mu\text{s}$ in the queue. The credit value goes from 516 to -568.

Appendix D

Additional simulation results

D.1 Sim 1: $\pm 10\%$ variation in send interval (using seed-set 1)

H1	H2	H3	H4	M1	M2	M3	M4	L1
996,363 63545 μs	1002,5877 6328 μs	1000,461 μs	1002,341 02306 μs	1000,292 29229 μs	999,621 μs	999,5779 9999 μs	1000,001 μs	1004,9055 2763 μs

Table D.1: Send interval average for each source.

The average send interval for stream H is 1000,438 μs , and for stream M the average send interval is 999,873 μs .

Frame size	Maximum delay H [μs]	Maximum delay M [μs]	Average delay H [μs]	Average delay M [μs]
142 B	1198,36	1463,36	596,926	750,521
242 B	1206,36	1471,36	605,322	758,931
342 B	1214,36	1479,36	613,465	767,37
442 B	1258,679	1487,36	621,916	775,574
542 B	1258,679	1495,36	630,549	784,329
642 B	1258,679	991,53	641,537	384,19
742 B	1283,025	831	651,978	233,982
842 B	1286,68	839	662,695	228,744
942 B	1302,68	847	674,916	231,011

1042 B	1318,679	788,78	687,202	236,685
1142 B	1340,639	803,892	701,522	245,274
1242 B	1364,639	794,64	716,53	253,485
1342 B	1393,559	834,85	738,855	292,529

Table D.2:Maximum delay and average delay times for Sim 1. Using send interval variation of $1000 \pm 10\%$, with seed-set 1.

D.2 Sim 3: $\pm 2\%$ variation in send interval (using seed-set 1)

H1	H2	H3	H4	M1	M2	M3	M4	L1
999,661 33866 μs	1000,2682 6826 μs	999,84415 584 μs	999,883 μs	1000,290 29029 μs	999,444 μs	1000,286 μs	999,89 μs	1000,0690 6906 μs

Table D.3: Send interval average for each source.

Frame size	Maximum delay H [μs]	Maximum delay M [μs]	Average delay H [μs]	Average delay M [μs]
142 B	835,36	768,36	439,17	393,67
242 B	843,36	776,36	447,46	402,18
342 B	851,36	784,36	455,64	410,76
442 B	859,36	792,36	464,34	419,21
542 B	908,68	829	472,55	428,68
642 B	924,68	679,68	482,2	258,52
742 B	924,68	683,68	493,67	238,83
842 B	944,1	608,783	504,62	242,62
942 B	966,36	593,04	517,49	246,36
1042 B	972,89	606,03	529,96	250,68

1142 B	963,55	637,58	544,29	256,06
1242 B	960,68	655,59	558,47	252,55
1342 B	995,8	711,6	574,91	270,45

Table D.4: Maximum delay and average delay times for Sim 3. Using a send interval variation of $1000 \pm 2\%$, with seed-set 1.

D.3 Sim 1 modified and Sim 5 utilization statistics

Frame size	Frames dequeued queue 6	Frames dequeued queue 5	Utilization queue 6	Utilization queue 5	Frames dequeued queue 4	Utilization queue 4
142 B	3996 (H sent 3999)	3997 (M sent 4000)	4,539456Mb/s (reserved: 4,544Mb/s)	4,540592Mb/s (reserved: 4,544Mb/s)	7251	89,448336Mb/s
242 B	3997 (H sent 3999)	3997 (M sent 4000)	7,738192Mb/s (reserved: 7,744Mb/s)	7,738192Mb/s (reserved: 7,744Mb/s)	6737	83,107632Mb/s
342 B	3997 (H sent 3999)	3997 (M sent 4000)	10,935792Mb/s (reserved: 10,944Mb/s)	10,935792Mb/s (reserved: 10,944Mb/s)	6222	76,754592Mb/s
442 B	3996 (H sent 3999)	3997 (M sent 4000)	14,129856Mb/s (reserved: 14,144Mb/s)	14,133392Mb/s (reserved: 14,144Mb/s)	5708	70,413888Mb/s
542 B	3996 (H sent 3999)	3997 (M sent 4000)	17,326656Mb/s (reserved: 17,344Mb/s)	17,330992Mb/s (reserved: 17,344Mb/s)	5194	64,073184Mb/s
642 B	3996 (H sent 3999)	3998 (M sent 4000)	20,523456Mb/s (reserved: 20,544Mb/s)	20,533728Mb/s (reserved: 20,5979667Mb/s)	4679	57,720144Mb/s
742 B	3996 (H sent 3999)	3999 (M sent 4000)	23,720256Mb/s (reserved: 23,744Mb/s)	23,738064Mb/s (reserved: 24,7502116Mb/s)	4164	51,367104Mb/s
842 B	3996 (H sent 3999)	3999 (M sent 4000)	26,917056Mb/s (reserved: 26,944Mb/s)	26,937264Mb/s (reserved: 29,0168474Mb/s)	3650	45,0264Mb/s
942 B	3996 (H sent 3999)	3999 (M sent 4000)	30,113856Mb/s (reserved: 30,144Mb/s)	30,136464Mb/s (reserved: 33,6084485Mb/s)	3135	38,67336Mb/s
1042 B	3996 (H sent 3999)	3999 (M sent 4000)	33,310656Mb/s (reserved: 33,344Mb/s)	33,335664Mb/s (reserved: 38,5800898Mb/s)	2621	32,332656Mb/s

1142 B	3996 (H sent 3999)	3999 (M sent 4000)	36,507456Mb/s (reserved: 36,544Mb/s)	36,534864Mb/s (reserved: 44,0021963Mb/s)	2106	25,979616Mb/s
1242 B	3996 (H sent 3999)	3999 (M sent 4000)	39,704256Mb/s (reserved: 39,744Mb/s)	39,734064Mb/s (reserved: 49,9670608Mb/s)	1592	19,638912Mb/s
1342 B	3996 (H sent 3999)	3999 (M sent 4000)	42,901056Mb/s (reserved: 42,944Mb/s)	42,933264Mb/s (reserved: 56,5975072Mb/s)	1077	13,285872Mb/s

Table D.5: Sim 5 utilization statistics.

Frame size	Frames dequeued queue 6	Frames dequeued queue 5	Utilization queue 6	Utilization queue 5	Frames dequeued queue 4	Utilization queue 4
142 B	3997 (H sent 3999)	3997 (M sent 4000)	4,540592Mb/s	4,540592Mb/s	1000	12,336Mb/s
242 B	3997 (H sent 3999)	3997 (M sent 4000)	7,738192Mb/s	7,738192Mb/s	1000	12,336Mb/s
342 B	3997 (H sent 3999)	3997 (M sent 4000)	10,935792Mb/s	10,935792Mb/s	1000	12,336Mb/s
442 B	3997 (H sent 3999)	3997 (M sent 4000)	14,133392Mb/s	14,133392Mb/s	1000	12,336Mb/s
542 B	3997 (H sent 3999)	3997 (M sent 4000)	17,330992Mb/s	17,330992Mb/s	1000	12,336Mb/s
642 B	3997 (H sent 3999)	3998 (M sent 4000)	20,528592Mb/s	20,533728Mb/s	1000	12,336Mb/s
742 B	3997 (H sent 3999)	3999 (M sent 4000)	23,726192Mb/s	23,738064Mb/s	1000	12,336Mb/s
842 B	3997 (H sent 3999)	3999 (M sent 4000)	26,923792Mb/s	26,937264Mb/s	1000	12,336Mb/s
942 B	3996 (H sent 3999)	3999 (M sent 4000)	30,113856Mb/s	30,136464Mb/s	1000	12,336Mb/s

1042 B	3996 (H sent 3999)	3999 (M sent 4000)	33,310656Mb/s	33,335664Mb/s	1000	12,336Mb/s
1142 B	3996 (H sent 3999)	3999 (M sent 4000)	36,507456Mb/s	36,534864Mb/s	1000	12,336Mb/s
1242 B	3996 (H sent 3999)	3999 (M sent 4000)	39,704256Mb/s	39,734064Mb/s	1000	12,336Mb/s
1342 B	3996 (H sent 3999)	3999 (M sent 4000)	42,901056Mb/s	42,933264Mb/s	1000	12,336Mb/s

Table D.6: Sim 1 modified utilization statistics.

D.4 Sim5 with increased bandwidth reservation values

Frame size	Maximum delay H [μs]	Maximum delay M [μs]	Reservation H	Reservation M	Average delay H [μs]	Average delay M [μs]
142 B	745,319	733,44	5,048888889 Mb/s	5,048888889 Mb/s	201,993	194,693
242 B	719,32	729,2	8,604444444 Mb/s	8,604444444 Mb/s	208,921	204,873
342 B	694,08	780,32	12,16 Mb/s	12,16 Mb/s	214,238	215,512
442 B	766,439	763,76	15,715555554 Mb/s	15,715555554 Mb/s	222,07	226,374
542 B	742,88	827,199	19,2711111109 Mb/s	19,2711111109 Mb/s	229,129	237,573

642 B	760,8	773,92	22,82666666 4 Mb/s	22,82666666 4 Mb/s	237,054	249,896
742 B	792,88	827,72	26,38222222 Mb/s	26,38222222 Mb/s	244,447	261,279
842 B	774,48	856,68	29,93777777 5 Mb/s	29,93777777 5 Mb/s	253,064	272,693
942 B	756,08	782,92	33,49333333 Mb/s	34,05870788 421124 Mb/s	261,374	279,812
1042 B	778,76	779,16	37,04888888 5 Mb/s	39,23943776 8900298 Mb/s	270,295	283,335
1142 B	769,8	717,48	40,60444444 Mb/s	44,96151499 0452568 Mb/s	278,554	283,286
1242 B	825,199	802,6	44,15999999 6 Mb/s	51,36095857 2479638 Mb/s	287,905	291,919

Table D.7: Sim 5 (default seed-set), using increased bandwidth reservation values, as shown in table.

D.5 Sim1 with preemption

Queues 5 and 6 are set to express, queue 4 is set to preemptive.

Frame size	Maximum delay H [μs]	Maximum delay M [μs]	Average delay H [μs]	Average delay M [μs]
142 B	1468,36	1221,679	702,171	518,952
242 B	1476,36	1237,679	710,888	530,653

342 B	1484,36	1253,679	719,602	542,61
442 B	1492,36	1269,679	728,323	555,533
542 B	1500,36	1285,68	737,082	569,037
642 B	1509,29	938,069	747,758	391,795
742 B	1557,657	749,868	757,526	222,158
842 B	1524,36	734,783	767,257	212,793
942 B	1532,36	744,68	777,039	211,675
1042 B	1540,36	760,68	787,82	211,672
1142 B	1548,36	767,68	799,062	211,774
1242 B	1556,36	684,68	810,011	213,796
1342 B	1577,32	708,16	828,464	226,187

Table 6.1: Sim 1 with preemption: streams H and M may preempt L traffic.

Appendix E

Queue time Calculations

The calculations shown above is based on Sim 1 in subsection 4.4.1. Stream H has a bandwidth reservation of 4,544 Mb/s, which gives an idleSlope of 4,544 Mb/s, and a sendSlope of -95,456 Mb/s. The transmission time for an H frame is 11,36 μ s, the transmission time for an L frame is 123,36 μ s, and the idle time between two consecutive transmissions is 0,96 μ s. Note that in this example I've added the idle time to the L dep time,

this is not technically correct, but I've done so to better show the credit value right before the h1 frame is forwarded.

The scheduling scenario for the stream H frames is the same as the worst-case scenario shown in Figure 4.2. All L frames arrive with 1ms in between. I vary the send interval for the H frames, first I set it to the lower bound of the send interval variation in Sim 1, 900 μ s. I show that by setting the send interval to the lower bound there occurs a buildup in queue times. I then set the send interval to the middle value, 1000 μ s and show that this causes the queue times to remain constant. And, increasing the send interval to the upper bound 1100 causes a gradual decrease in queue times.

Event	Time [μ s]	Buffer	Delay [μ s]	Credit
h1,h2,h3,h4 arr	0	h1,h2,h3,h4		0
h4 dep	11,36	h1,h2,h3	11,36	-1084,38016
cr = 0	250			
h3 dep	261,36	h1,h2	261,36	-1084,38016
cr = 0	500			
h2 dep	511,36	h1	511,36	-1084,38016
L arr	749			
cr = 0	750			
L dep	873,32			560,36608
h1 dep	884,68		884,68	-524,01408
h1,h2,h3,h4 arr	900	h1,h2,h3,h4		
cr = 0	1000			
h4 dep	1011,36	h1,h2,h3	111,36	-1084,38016
cr = 0	1250			
h3 dep	1261,36	h1,h2	361,36	-1084,38016
cr = 0	1500			
h2 dep	1511,36	h1	611,36	-1084,38016
L arr	1749			
cr = 0	1750			
h1,h2,h3,h4 arr	1800	h1,h2,h3,h4,h1		
L dep	1873,32			560,36608
h1 dep	1884,68	h1,h2,h3,h4	984,68	-524,01408
cr = 0	2000			

h4 dep	2011,36	h1,h2,h3	211,36	-1084,38016
cr = 0	2250			
h3 dep	2261,36	h1,h2	461,36	-1084,38016
cr = 0	2500			
h2 dep	2511,36	h1	711,36	-1084,38016
h1,h2,h3,h4 arr	2700	h1,h2,h3,h4,h1		
L arr	2749			
cr = 0	2750			
L dep	2873,32			560,36608
h1 dep	2884,68	h1,h2,h3,h4	1084,68	-524,01408
cr = 0	3000			
h4 dep	3011,36	h1,h2,h3	311,36	-1084,38016
cr = 0	3250			
h3 dep	3261,36	h1,h2	561,36	-1084,38016
cr = 0	3500			
h2 dep	3511,36	h1	811,36	-1084,38016
h1,h2,h3,h4 arr	3600	h1,h2,h3,h4,h1		
L arr	3749			
cr = 0	3750			
L dep	3873,32			560,36608
h1 dep	3884,68	h1,h2,h3,h4	1184,68	-524,01408
cr = 0	4000			
h4 dep	4011,36	h1,h2,h3	411,36	-1084,38016
cr = 0	4250			
h3 dep	4261,36	h1,h2	661,36	-1084,38016
cr = 0	4500			
h2 dep	4511,36	h1	911,36	-1084,38016
h1,h2,h3,h4 arr	4600	h1,h2,h3,h4,h1		
L arr	4749			
cr = 0	4750			
L dep	4873,32			560,36608
h1 dep	4884,68	h1,h2,h3,h4	1284,68	-524,01408
cr = 0	5000			
h4 dep	5011,36	h1,h2,h3	411,36	-1084,38016
cr = 0	5250			

h3 dep	5261,36	h1,h2	661,36	-1084,38016
cr = 0	5500			
h2 dep	5511,36	h1	911,36	
h1,h2,h3,h4 arr	5600	h1,h2,h3,h4,h1		
L arr	5749			
cr = 0	5750			
L dep	5873,32			560,36608
h1 dep	5884,68	h1,h2,h3,h4	1284,68	-524,01408
cr = 0	6000			
h4 dep	6011,36	h1,h2,h3	411,36	-1084,38016
cr = 0	6250			
h3 dep	6261,36	h1,h2	661,36	-1084,38016
cr = 0	6500			
h2 dep	6511,36	h1	911,36	-1084,38016
h1,h2,h3,h4 arr	6600	h1,h2,h3,h4,h1		
L arr	6749			
cr = 0	6750			
L dep	6873,32			560,36608
h1 dep	6884,68	h1,h2,h3,h4	1284,68	-524,01408
cr = 0	7000			
h4 dep	7011,36	h1,h2,h3	411,36	-1084,38016
cr = 0	7250			
h3 dep	7261,36	h1,h2	661,36	-1084,38016
cr = 0	7500			
h2 dep	7511,36	h1	911,36	-1084,38016
h1,h2,h3,h4 arr	7700	h1,h2,h3,h4,h1		
L arr	7749			
cr = 0	7750			
L dep	7873,32			560,36608
h1 dep	7884,68	h1,h2,h3,h4	1284,68	-524,01408
cr = 0	8000			
h4 dep	8011,36	h1,h2,h3	311,36	-1084,38016
cr = 0	8250			
h3 dep	8261,36	h1,h2	561,36	-1084,38016
cr = 0	8500			

h2 dep	8511,36	h1	811,36	-1084,38016
L arr	8749			
cr = 0	8750			
h1,h2,h3,h4 arr	8800	h1,h2,h3,h4,h1		
L dep	8873,32			560,36608
h1 dep	8884,68	h1,h2,h3,h4	1184,68	-524,01408
cr = 0	9000			
h4 dep	9011,36	h1,h2,h3	211,36	-1084,38016
cr = 0	9250			
h3 dep	9261,36	h1,h2	461,36	-1084,38016
cr = 0	9500			
h2 dep	9511,36	h1	711,36	-1084,38016
L arr	9749			
cr = 0	9750			
L dep	9873,32			560,36608
h1 dep	9884,68		1084,68	-524,01408
h1,h2,h3,h4 arr	9900	h1,h2,h3,h4		
cr = 0	10000			
h4 dep	10011,36	h1,h2,h3	111,36	-1084,38016
cr = 0	10250			
h3 dep	10261,36	h1,h2	361,36	-1084,38016
cr = 0	10500			
h2 dep	10511,36	h1	611,36	-1084,38016
L arr	10749			
cr = 0	10750			
L dep	10873,32			560,36608
h1 dep	10884,68		984,68	-524,01408
cr = 0	11000			
h1,h2,h3,h4 arr	11000	h1,h2,h3,h4		
h4 dep	11011,36	h1,h2,h3	11,36	-1084,38016
cr = 0	11250			
h3 dep	11261,36	h1,h2	261,36	-1084,38016
cr = 0	11500			
h2 dep	11511,36	h1	511,36	-1084,38016
L arr	11749			

cr = 0	11750			
L dep	11873,32			560,36608
h1 dep	11884,68		884,68	-524,01408
cr = 0	12000			
h1,h2,h3,h4 arr	12100			
h4 dep	12111,36	h1,h2,h3	11,36	-1084,38016
cr = 0	12350			
h3 dep	12361,36	h1,h2	261,36	-1084,38016
cr = 0	12600			
h2 dep	12611,36	h1	511,36	-1084,38016
L arr	12749			
cr = 0	12850			
L dep	12873,32			105,96608
h1 dep	12884,68		784,68	-978,41408
cr = 0	13100			
h1,h2,h3,h4 arr	13200	h1,h2,h3,h4		0
h4 dep	13211,36	h1,h2,h3	11,36	-1084,38016
cr = 0	13450			
h3 dep	13461,36	h1,h2	261,36	-1084,38016
cr = 0	13700			
h2 dep	13711,36	h1	511,36	-1084,38016
L arr	13749			
L dep	13872,36			
cr = 0	13950			
h1 dep	13961,36		761,36	-1084,38016
cr = 0	14200			
h1,h2,h3,h4 arr	14300	h1,h2,h3,h4		
h4 dep	14311,36	h1,h2,h3	11,36	-1084,38016
cr = 0	14550			
h3 dep	14561,36	h1,h2	261,36	-1084,38016
L arr	14749			
cr = 0	14800			
L dep	14873,32			333,16608
h2 dep	14884,68	h1	584,68	-751,21408
cr = 0	15050			

h1 dep	15061,36		761,36	-1084,38016
cr = 0	15300			
h1,h2,h3,h4 arr	15400	h1,h2,h3,h4		

Figure E.1: Queue times calculations: build up, static, decrease.