Simulation of Instability in Time-Sensitive Networks with Regulators and Imperfect Clocks

Master Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Guillermo Aguirre Rodrigo

In partial fulfillment
of the requirements for the master in
TELECOMMUNICATIONS **ENGINEERING**

Advisor: Prof. Jean-Yves Le Boudec, Ludovic Thomas and Prof. Xavier Hesselbach Serra
Barcelona, Date 01/07/2020

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona telecos BCN EPFL

Abstract

Regulators are used in time-sensitive networks in order to reshape traffic reducing the burstiness generated by the aggregation systems within the network. Regulators, in terms of the IEEE TSN group are defined as Asynchronous Traffic Shaping or what in the literature is known as interleaved regulators. These regulators are placed before a multiplexing stage, shaping the outgoing traffic and cancelling the increased burstiness, allowing calculations of the worst-case delay to be performed. One of the main properties studied for regulators implies that they do not increase the worst-case delay in the network. So far, all the studies were performed assuming ideal clocks within the network. However, a recently published paper underlines the importance of taking into account non-ideal clocks while studying interleaved regulators properties, providing a theoretical proof that under certain circumstances, interleaved regulators can lead to system instability due to the unbounded delay through them. This thesis intends to continue with this study, providing a simulation proof that shows instability when interleaved regulators are simulated assuming non-ideal clocks. This is carried out in the network simulator ns-3. Along with the different developed modules, two are highlighted: local time and ATS modules. The former introduces the notion of local times in ns-3, whereas the latter implements regulators functionality in ns-3, following the implementation proposed by the IEEE TSN.

Revision history and approval record

Revision	Date	Purpose
0	28/05/2020	Document Creation
1	27/06/2020	Document Revision

Written by: Guillermo Aguirre		Reviewed and approved by: Ludovic Thomas	
Date	July 2020	Date	July 2020
Position	Project Author	Position	Project Supervisor

Acknowledgements

First of all, I wish to thank my supervisor in the EPFL, Ludovic Thomas and Jean-Yves Le Boudec for helping me throughout the project and giving me the opportunity to carry out the thesis in their laboratory. Also, thanks to my supervisor in the UPC Xavier Hesselbach Serra for helping me with all the administrative work. Finally, thanks to Carmen Sirés for helping me to review the thesis.
On a personal level, I would like to thank the Swiss family that adopted me during these confinement times and treated me as if I were in my home.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Over the last decade, Time Sensitive Networks (TSN) have gained a relevant importance in the network communication field, and researches have been conducted by the academic and industry fields. Real-time applications and distributed safety-critical applications in industries, such as automotive [25], manufacturing [32] and avionics, have underlined the relevance of deterministic communications. Even the 5G technology aims to integrate TSN within Industry 4.0 [12]. Driven by the industry needs, efforts to standardize real-time communications over Ethernet have been made, resulting in the IEEE 802.1 Time-sensitive Networking (TSN) Task Group (TG). Time-sensitive networks, according to IEEE organization, is a set of standards which attempt to guarantee packet transport, low packet loss and bounded low latency, providing new levels of connectivity, optimization and cost saving with seamless reconfiguration and redundancy.

A general problem in networks is that aggregate buffering and scheduling increases the burstiness of the flows that share the same system resources. Imagine different flows perfectly shaped with different rates and burst sizes, this is, the interval time between packets is perfectly defined. All these flows share the same output port of a router and the output bandwidth of the link is much bigger that the total rate of the input flows. Even in this situation, sudden build-up of packets at the output port, due to some of the flows arriving at the same time, can drastically increase the jitter of some flows, resulting in the decrease of the interval time between packets of the same flow and the increase of the so-called burstiness. The problem can be exacerbated if an already bursty traffic shares again another aggregation system, increasing even more the burstiness of the traffic as studied in [3][8]. Burstiness of every flow increases at every hop as a function of other flows burstiness, leading to the burstiness cascade problem. The creation of bursts within the network makes the computation of upper bounds for the latency difficult, and sometimes can lead to system instability with unbounded delays.

To cope with this problem, TSN propose to reshape traffic by means of regulators. Regulators are hardware elements placed at the output ports of network devices that reshape traffic, forcing flows to have specific burst sizes. They remove the burstiness created by the aggregate scheduling systems, bringing back the original burst size of the flows. Regulators come in two types; per-flow regulators and interleaved regulators. Within this project, we stay focused in the

Interleaved Regulator (IR), which can also be called Asynchronous Traffic Shaping (ATS).

Interleaved regulators have been widely studied in different papers, like [6] and [5], and their properties and functions are well known. However, the majority of the studies conducted in this field have been done assuming ideal clocks within the networks. Nothing could be further from the truth, network devices use local clocks, which are imperfect clocks. In order to deal with clock deviations, TSN propose two types of networks, synchronized and non-synchronized networks. In synchronized networks, clocks deviations are kept within some bounds, using a synchronization protocol, such as Network Time Protocol (NTP) [22] or Precicion Time Protocol (PTP) [11]. The degree at which synchronized networks keep bounds classifies them into two levels, ”tightly” synchronized and ”loosely” synchronized. In non-synchronized networks, clocks run independently one from each other.

The research carried out by Ludovic Thomas and Jean-Yves Le Boudec in [20], studies regulators under different time synchronization issues and non-ideal clocks. Among the various conclusions achieved, one acquires a greater importance, concerning interleaved regulators within synchronized networks and non-ideal clocks. A theoretical proof is presented, showing that the interleaved regulator, under a particular time model and network conditions, leads to system instability and unbounded delay.

In this project, a natural step from the theory to the simulation is proposed. We continue the research carried out in [20], and more precisely, what concerns interleaved regulators and synchronized networks, providing a simulation proof that confirms the theoretical proof.

The project has been carried out in the network simulator ns-3. As an intermediate step to achieve the final simulation, time notions have been introduced in the simulator. The simulation of different clocks in the network simulator is an important progress and interesting feature for ns-3. Because of that, a design of an independent module that includes the basic functionalities to implement local clocks in ns-3, is considered. The subsequently upload of the clock module to the ns-3 source code has been established as another important goal of the thesis.

1.1 Thesis contribution

The main contributions of this thesis are:

- The concept of different time notions and clocks is introduced in ns-3. As it stands, ns-3 does not support the simulation of different clocks within the nodes of the simulation. A clear and open interface that allows to introduce clocks in the already existing code of the simulator is presented. We believe that this new feature extends the capabilities of the simulator, allowing a large variety of new scenarios.

- A simulation framework for interleaved regulators and, with minimum changes, for per-fow regulators in ns-3 is provided.

- By the use of interleaved regulators and clock notions in ns-3, a simulation proof that confirms the theoretical studies carried out with interleaved regulators and synchronized

networks is provided, verifying the system instability and unbounded delay.

- An extension of the proof is proposed, both mathematically and by simulation, making the proof more meaningful and suitable for a wider variety of networks.

1.2 Thesis outline

The master thesis is structured as follows:

- **Chapter 1** Introduces the general topic of the thesis, as well as the objectives proposed.
- **Chapter 2** Provides the necessary background to understand and follow the main topics of the thesis, as well as the explanation of the theoretical proof that is simulated.
- **Chapter 3** Describes the main modules developed in ns-3 to achieve the final simulation. The chapter includes, the explanation of the module developed to introduce the notion of local times in ns-3, the ATS module that allows to simulate interleaved regulators in ns-3 and finally the scenario model proposed for the simulation.
- **Chapter 4** Presents the main results of the simulation, as well as the extension proposed for the proof.
- **Chapter 5** Presents the conclusion and future development.
- **Chapter 6** Explains the budget of the thesis.

1.3 Gantt Diagram

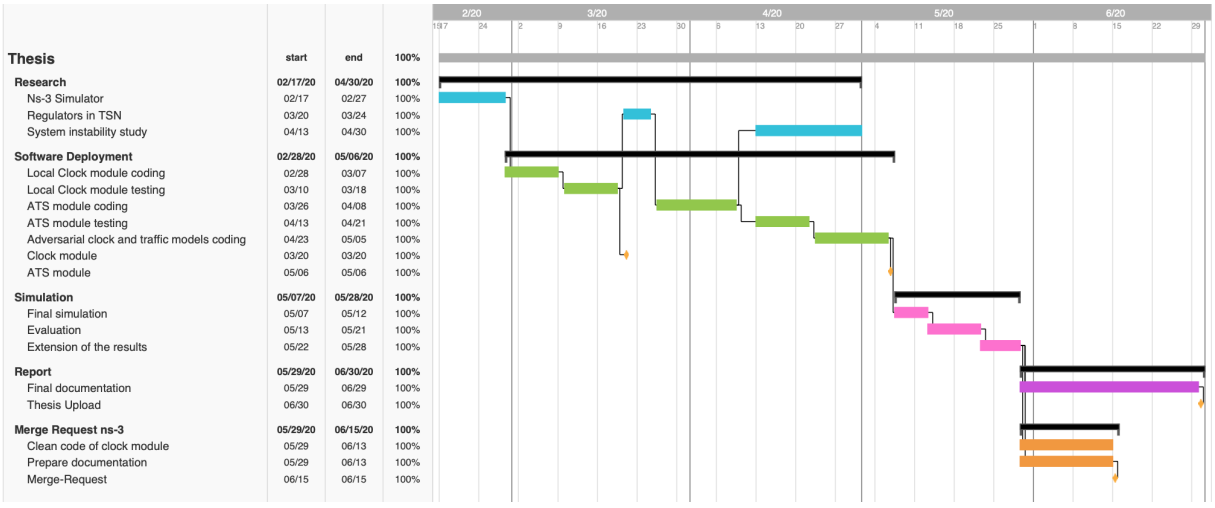


Figure 1.1: Gantt Diagram

Chapter 2

State of the art

This chapter introduces the background information needed to understand the main parts of the project, as well as the state of the art of the used technologies. Section 2.1 provides a brief and intuitive explanation of the main concepts of network calculus, in order to better understand how delay bounds are computed in a network. Section 2.2, introduces the concepts of regulators and their main properties. Also, the state of the art is explained in section 2.2.1 in terms of implementation of interleaved regulators. Section 2.3, introduces the main results of the research [20], concerning regulators and time synchronization issues under non-ideal clocks. Finally, section 2.4, explains the main concepts behind the ns-3 simulator.

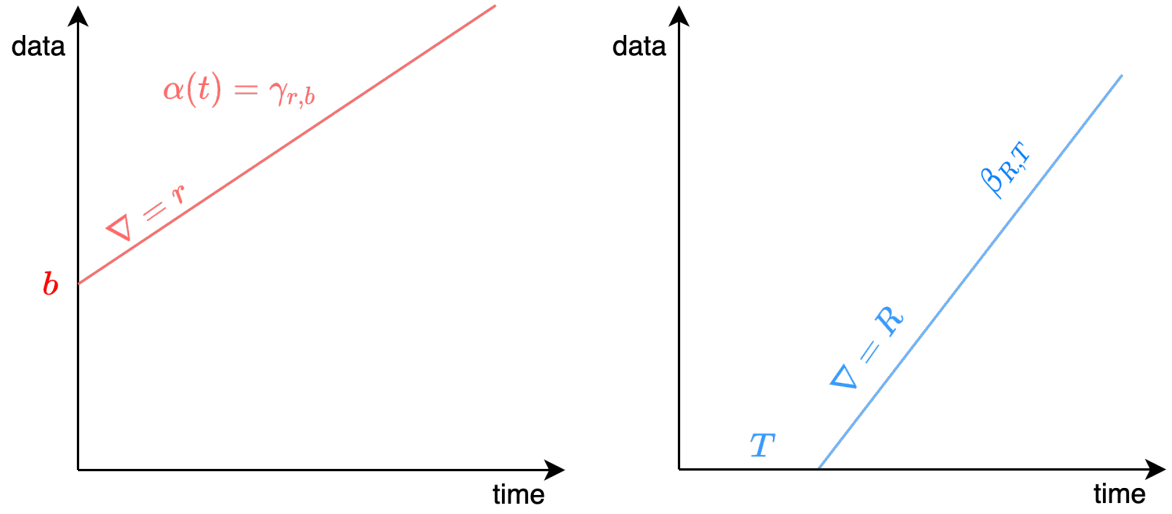
2.1 Network calculus basic concepts

In order to better understand some of the concepts used along the thesis and how delays bounds are calculated in networks, an intuitive explanation is given with the main concepts of network calculus. Network calculus is a mathematical framework that allows to analyse performance guarantees in computer networks. It's a "system theory" based on Min-Plus algebra and maintains some similarities with the traditional system theory applied to circuits and signals, such as the convolution operation. Networks calculus provides a framework to model flows and network nodes.

To provide warranty services within the network we need to constrain the flows, to this aim arrival curves are proposed. In the case of flows shaped by a token bucket, the rate r and burst size b are constrained by $\gamma_{r,b}(t) = rt + b$. This arrival curve upper-bounds the traffic that source sends by sending at once b bits but constraining to r b/s the output of the source, as seen in figure 2.1a.

As well as flows are upper-bounded, the service provided by network nodes need to be modelled. The network nodes are also called schedulers, the FIFO system being the most common and used scheduler. The service provided is modelled using the concept of services curves. Imagine a network node that provides a FIFO queue for all the flows of the network, each FIFO system serves packets at a rate R and time T (e.g. with T due to a processing delay

or another fixed delay introduced by the FIFO). The service curve can be modelled by $\beta_{R,T}$, graphically shown in figure 2.1b.



(a) Arrival curve

(b) Service curve

Figure 2.1: Arrival and service curves

Network calculus studies the different arrival curves and services curves allowing to calculate different bounds within the network. In this section we focus on three bounds that can be easily derived using figure 2.2. In this example, the buffer size of the FIFO queue, the max delay for any frame in the FIFO system, as well as the output bounds of the incoming flow can be established. As it can be seen in figure 2.2, the buffer required for the FIFO system is equal to the vertical distance between the time T and the intersection with the arrival curve. The maximum delay in the FIFO system, which corresponds to the horizontal distance between the burst b and the intersection with the service curve, is equal to b/R if $R \geq r$, if not is infinitive. Moreover, the burstiness increase of the output flow can be computed. The new arrival curve of the output flow, represented by the green line, has an increased burstiness equal to $b' = b + rT$.

This model can be extended to other flow patterns and services curves in a network with more nodes. In this section basic concepts are explained in order to have an intuitive knowledge of how bounds are calculated. As I stated before, when speaking about TSN we want to provide upper bounds of the delay that the traffic will have along the network. However, as we can see, if the traffic burstiness increases along the network in each hop, the calculation tends to be more difficult and, in some cases, leads to an unbounded delay and therefore to system instability. As we will seen in the next section, regulators remove the burstiness created by the aggregation systems, such as the FIFO queue.

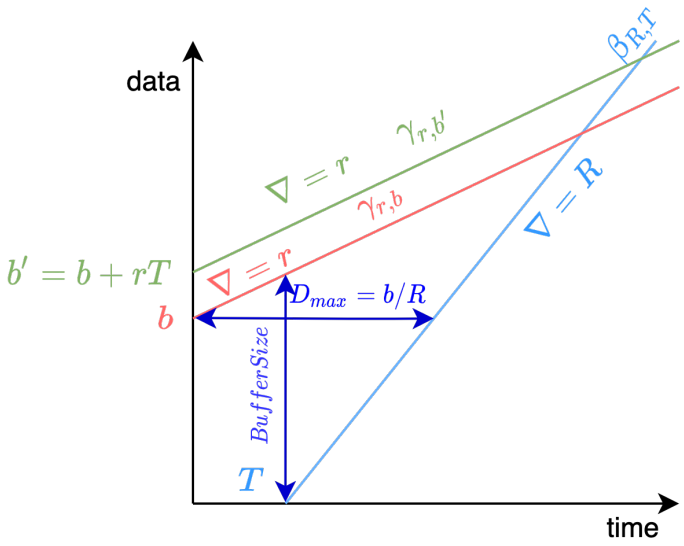


Figure 2.2: Bounds on the buffer and the maximum delay

2.2 Regulators

Regulators are hardware elements placed at the output ports of network devices. Their aim is to remove the burstiness created by the iterations of different flows reshaping the traffic, as seen in figure 2.3, allowing the calculation for delay bounds.

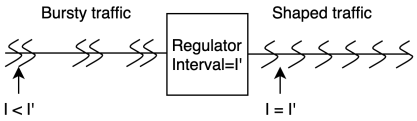


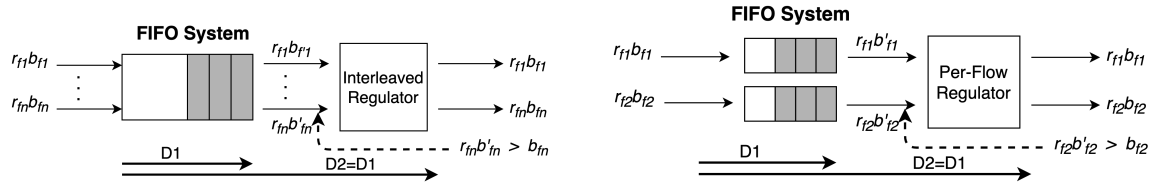
Figure 2.3: Bursty traffic reshaped by the regulator

Traffic regulators come in two flavors, per-flow regulators (PFR) and interleaved regulators (IR). IR are called Asynchronous Traffic Shaping (ATS) too, as stated in 802.1Qcr draft [4]. Per flow regulators (figure 2.4b), process each flow individually and require one FIFO queue per flow, whereas the interleaved regulators (figure 2.4a), process flow aggregates and require one FIFO queue per input output port and traffic class.

In both types of regulators, shaping parameters are established based on flows. This is, each flow is shaped with its own parameters. Normally, the shaping is done using a token bucket machine with a burst b and rate r parameters. This token bucket machine is comparable to the Linux Token Bucket Filter [17]. The regulator forces a policy on the flow, packets that violate the contract are delayed. Applying the token bucket concepts, no more than $b + rt$ bits can be released in a time t. If properly tuned, flows at the output of the regulators will have the same burtiness than the original flows. However, the shaping mechanism used is not bounded to a

token bucket shaping algorithm, opening up the possibility for different implementations. Due to the state information needed per flow, which can increase to thousands of flows, other solutions such as Regulating Schedulers [13] or Linear Quadratic Algorithms (LQR) are proposed.

As it's been mentioned before, the main difference between the per-flow regulator and interleaved regulator is that the former uses one FIFO queue system per flow whereas the later uses one FIFO system for all the aggregated flows. Interleaved regulator examines the packets at the head of its FIFO system and it is released as soon as it does not violate the regulation constrains of the flow. Packets that are not at the head of the queue are not considered, therefore a packet could be delayed because of regulation constrains of the flow, or because the packet at the head of the queue is delayed. When computing delay bounds, this introduced delay has to be taken into account.



(a) Shaping for free in IR

(b) Shaping for free in PFR

Figure 2.4: Shaping for free property, from [20]

One of the key properties of regulators that allows to calculate bounded delay, is what can be called "shaping for free property". This property, widely studied in [6], shows that any regulator placed at the output of any arbitrary system that is FIFO for the flow of interest or FIFO for the aggregate flows, does not increase the worst-case delay of the flow, as seen in figures 2.4a and 2.4b . In both cases, we have flows entering the FIFO system that are characterized by the parameters rate and burst $(r_{f,n}, b_{f,n})$. The flows at the output of the FIFO system have an increased burtiness $(b'_{f,n})$. Regulators reshape the flows to the original parameters $(r_{f,n}, b_{f,n})$ if they have been tuned properly. If this holds, the shaping property holds. The worst-case delay of the FIFO system is characterized by *D1*. Shaping for free states that *D2*, which is the worst-case delay at the output of the regulator, is equal to *D1*. This means that any packet that gets delayed excessively in the FIFO queue (worst-case delay), will be immediately released, without introducing any extra delay. This is inherent to both types of regulators.

Appendixes D.1 and D.3, provide some intuitive graphs about the general functioning of IR or ATS. These appendices explain the validation process for the IR introduced in ns-3. However, some intuitive explanations that could help to understand the performance of regulators are given.

2.2.1 Asynchronous traffic shaping (ATS)

In TSN, interleaved regulators are preferred over per-flow regulators. TSN propose class-based networks, where flows are classified within some classes and processes in a FIFO manner.

Each class contains one FIFO, therefore the behavioural model corresponds to the interleaved regulator where one IR is placed in each class.

Currently, the specific implementation of an interleaved regulator in a bridge is still an open discussion. The basic requirements are outlined in the draft 802-1Qcr [4], which is in the D2.3 version. It gathers all the procedures and managed objects for bridges and end station to perform asynchronous traffic shaping over full duplex links and constant bit rates. The asynchronous terms indicate that interleaved regulators do not need to be synchronized within the network (i.e. clocks in different devices do not need to be synchronized), in contrast with the time-aware shapers [2] that need a tight synchronization.

The second phase of the project was fully used to implement an asynchronous traffic shaping model or IR model in ns-3. Therefore, the draft 802-1Qcr has been used as a model to map the requirements of ATS into the ns-3 architecture. In this section, main characteristics of the draft 802-1Qcr used for the project are explained.

2.2.1.1 Forwarding process

We focus only on the model that supports ATS implementation, as depicted in figure 2.5. The figure represents the forwarding process on the output port of a bridge. It represents the TSN proposal in terms of per-class queuing, scheduling and shaping. Outgoing frames are classified by the stream filters depending on a set of parameters (i.e. priority level, stream handle value, etc). Each frame is associated with one stream filter and some policies can be applied to each frame. Each stream filter has one stream gate and one ATS scheduler attached to it. The stream gate basically discards frames whose reception time contradicts a given time schedule. As it can be seen, the policing action are taken by the stream filters and stream gates, whereas the shaping functionality is done by the ATS scheduler.

ATS schedulers assign eligibility times in non-decreasing order to frames, which are then used for traffic regulation by the ATS transmission selection algorithm. Eligibility times are computed based on slightly modified token bucket state machine. Parameters, such as Committed Burst Size (CBS) and Committed Information Rate (CIR), which are the parameters rate r and burst b for each flow, are used for shaping, in such a way that different flows can be shaped with different parameters. For a given queue that supports ATS transmission selection, the ATS transmission algorithm would determine the frames eligible to be transmitted. A frame is eligible for transmission if its eligibility time is equal or less than the current time shown by the local clock of the device. As it can be appreciated, the ATS scheduler performs the computation part of the shaping, however, the complete shaping operation is done by the combination with the ATS transmission algorithm.

ATS schedulers at the same time, are organized into ATS scheduler groups. There is one ATS scheduler group per input port and traffic class. ATS schedulers that perform operations for the same input port and traffic class, belong to the same ATS scheduler group. This allows to assign eligibility times in a non-decreasing order for ATS schedulers that belong to the same group, queuing frames of the same group in a FIFO manner. Thus, this shaping mechanism approaches to the one commented in section 2.2 for IR. All the flows that belong to the same

input port, traffic class and output port share the same FIFO system, in this case, forced by the group eligibility time.

The forwarding process provides one or more queues for each bridge port, each of them attached to a different traffic class. The queue system in general is a FIFO system, however, the draft is not appended to a specific queue. A queue in this context is a set of frames awaiting to be transmitted on a given bridge port. As we can see in the in figure 2.5, it's not the ATS transmission algorithm who sends the frames, it just determines which frames are available to be transmitted. Other transmission algorithms, such as fixed priority queue or a credit-base shaper retrieve the frames from each class.

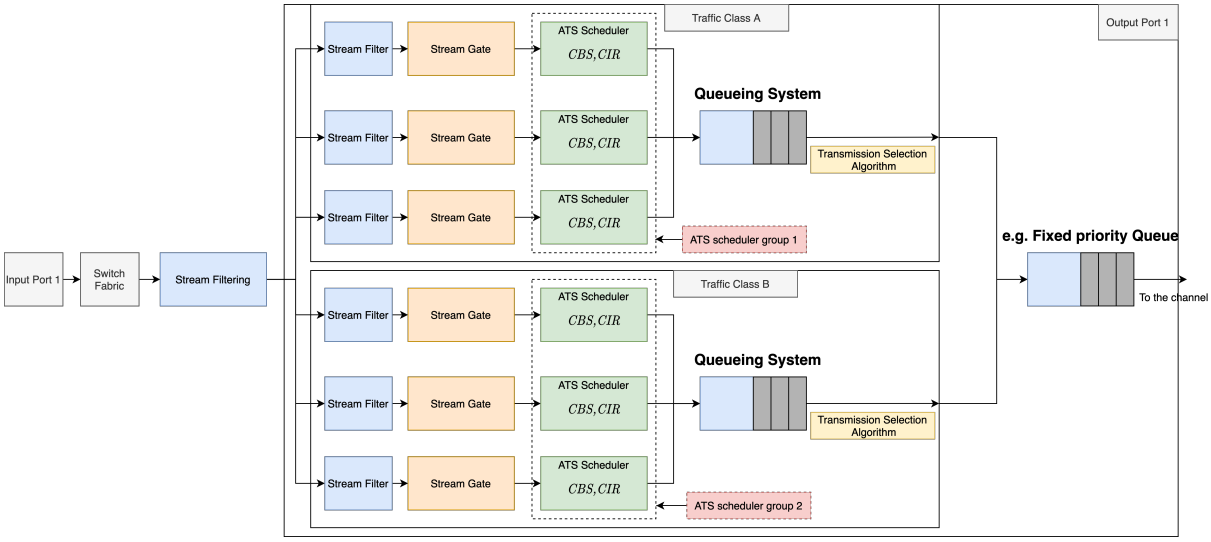


Figure 2.5: ATS bridge architecture and forwarding process

As a part of this project, we are not interested in all the building blocks that take part on the forwarding process of an ATS bridge. Components where policies are applied, such as stream filters and stream gates are not taken into account for the development and the purpose of the project. Nevertheless, the implementation of ATS in ns-3 has been done modular, opening up the possibility to include these features. The specific implementation of ATS in ns-3 is detailed in section 3.2.

2.3 System instability study

Burstiness and the so called burstiness cascade problem, arise only under certain traffic patterns conditions and network topologies. However, the existence of a proof that showed an unbounded delay, forced to introduce the concept of regulators. The certification process for TSN network equipment required a strong validation process due to the safety critical nature of TSN applications. The evidence of an adversarial case that can trigger instability on the system is enough to reconsider some aspects of the technology. It is in this context, where the importance of the proofs presented by Ludovic Thomas and Jean-Yves Le Boudec in [20] resides. This paper presents a theory for adding clock non-idealities into network calculus and apply them

to time-sensitive networks with regulators, achieving meaningful results in terms of systems instability for regulators.

In this section, the main results of [20] are highlighted. Also, in section 2.3.1 some of the main concepts related with the time model used in the proof are explained. Finally, section 2.3.2 contains the proof proposition for non-adapted IRs in synchronized networks.

To bring some context, a simple example that could lead to instability in synchronized and non-synchronized networks is proposed in [20, Section 1]. Consider a flow with a rate r and a burst b. When a packet is enqueued (e.g. FIFO system), the flow can become more bursty if other flows are in the queue. To reinforce the b again, an interleaved regulator (IR) is placed. This IR reinforces the flow burst b, and allows to bound the delay. However, imagine that the clock of the IR runs slower than the clock at the source. The true value of r implemented at the IR is less that the one in the source. This would lead to and additional delay introduce by the IR, that could generate the system instability and preclude the calculation of the worst-case delay. This simple example shows that if a regulator clock suffers non-idealities, the delay could be unbounded and lead to system instability.

Regulator studies have been done assuming ideal clocks. Basic properties, such as “shaping for free property”, have been studied assuming perfect clocks. Nonetheless, clocks implemented in real networks are non-ideal. They suffer from non-idealities related to the oscillators, temperature and many other variants gathered in [10]. The study [20] introduces a new set of tools to add clock non-idealitites in the framework of network calculus, allowing them to study theoretically the behaviour of regulators under different conditions. The study considers per-flow regulators and interleaved regulator or ATS, under synchronized networks and non-synchronized networks, taking into account clock non-idealities. The main conclusion achieved by [20] is as follows:

- Under ”loosely” synchronized and non-synchronized networks, per-flow regulators, as well as interleaved regulators, show system instability if clocks are non-adapted. Not adapted refers to the fact that parameters *r, b* configured in the regulator for each flow, do not account for possible clock deviations.

- Under ”tightly” synchronized networks, per-flow regulators do not show any system instability. However, interleaved regulators show instability if non-adapted.

The study concludes that interleaved regulators under any degree of synchronization show instability. The result obtained in non-synchronized networks is expected and actually the ongoing draft 802.1Qcr [4] takes into account these possible variations. However, the surprisingly result comes when IR are studied within synchronized networks. It does not matter which degree of synchronization is used, the IR show instability and an unbounded delay.

The simulation in ns-3 focuses on this last statement, IR under synchronized networks, moving from the theoretical proof towards a simulation proof. The underlying principle of the simulation is based on [20, Section 7.2-Appendix 10]. In order to better understand the proof and the simulation, the following section gives a brief explanation of the essential concepts.

2.3.1 Time model

This section introduces the clock model that will be later used in the simulation. [20, Section 4] introduces a general time model that captures clock non-idealities in synchronized and non-synchronized networks. In this section, we introduce the basic characteristics of the time model used for synchronized networks in order to understand the values used for the simulation.

Clocks are modelled using a relative time-function between a pair of clocks $\mathcal{H}_i$ and $\mathcal{H}_j$, denoted as $d_{j\to i}(t)$. This is the value that clock $\mathcal{H}_i$ would have when clock $\mathcal{H}_j$ shows t, if both have infinite precision, as shown in figure 2.6. The dotted line represents the perfect relative time-function between both clocks, where both show the same value at time t. The continuous and strictly increasing function represents a more natural relative time-function between $\mathcal{H}_i$ and $\mathcal{H}_j$. Due to the stochastic behaviour of clocks, the model focuses on bounding the evolution of the time relative time-function, however, it is impossible to provide perfect bounded values.

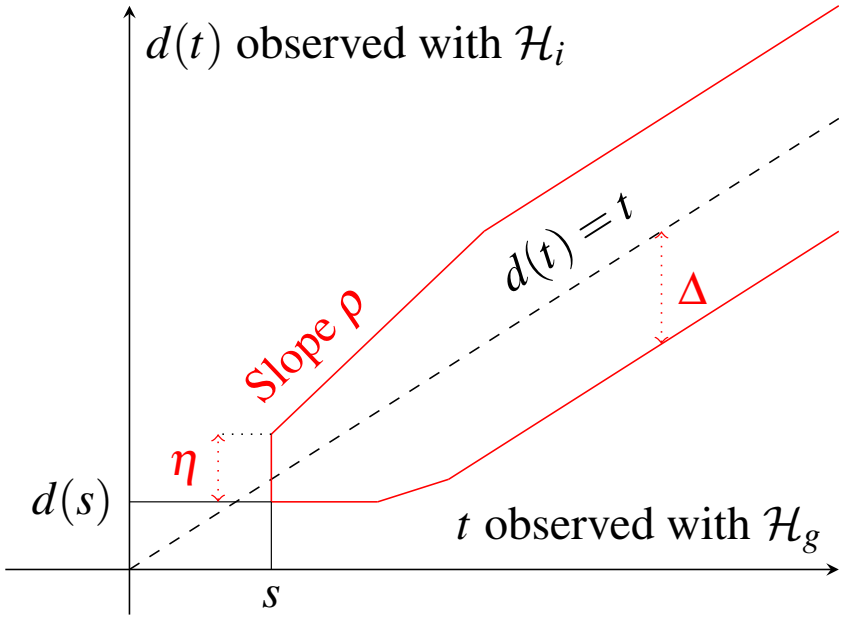


Figure 2.6: Evolution and envelope of d(t) from [20, Section 4.2]

To bound the relative time-function in synchronized networks we focus on three main parameters:

- η is value that bound the peak-to-peak jitter due to a high frequency noise signal in the clock.
- ρ value denotes the upper bounding of frequency offset between clocks and already takes into account the temperature variable.
- Δ value denotes the synchronization protocol precision.

[20, Section 4] states that any evolution of a relative time-function between two clocks within a

synchronized network can be bounded by the following two equations:

$$\forall t,s \quad \frac{1}{\rho}(t-s-\eta) \leq d(t)-d(s) \leq \rho(t-s)+\eta \tag{2.1}$$

$$\forall t, |d(t)-t| \leq \Delta \tag{2.2}$$

Equation 2.1 describes the evolution of the bounded relative time-function, where $d(t) = d_{g\to i}(t)$. This bounded evolution can be applied to clocks within synchronized networks, but also to clocks within non-synchronized networks. Because we have to take into consideration that the clocks are within synchronized networks, we define the time-error bound, which is described by the equation 2.2. The time-error bound, defined as $d_{g\to i}(t)-t$ is bounded by the precision of the protocol used for synchronization, denoted as Δ. Δ values, can be varied from $\Delta = 1\mu$s for tightly synchronized networks[1, Anex B.3] to $\Delta = 100m$s for loosely synchronized networks using the NTP protocol [22].

Using these three parameters we can create a bounded envelope for the relative time-function as shown in 2.6. The future evolution of any relative time-function between a pair of clocks must remain in the envelope.

2.3.2 Proof proposition

The following table of notation is presented in order to facilitate the proof understanding.

Table 2.1: Table of Notation

$\mathcal{H}_j$	$\triangleq$	Clock of a source
$\mathcal{H}_{IR}$	$\triangleq$	Clock of the IR
$\mathcal{H}_{FIFO}$	$\triangleq$	Clock of the FIFO system
$d_{IR\to j}(t)$	$\triangleq$	Relative time-function between $\mathcal{H}_{IR}$ and $\mathcal{H}_j$
t	$\triangleq$	The measure of a time instant
f_j	$\triangleq$	Flow output by source j
s_1	$\triangleq$	Frequency offset variation between $\mathcal{H}_{IR}$ and $\mathcal{H}_j$
I	$\triangleq$	Interval time between packets of the same flow in the same period
τ	$\triangleq$	Period duration
k	$\triangleq$	Period number
$d_j(x_j)+k\tau$	$\triangleq$	Instant of time at which source j send the first packet in period k in local time
$x_j+k\tau$	$\triangleq$	Instant of time at which source j send the first packet in period k in global time
$A^i_{j,k}$	$\triangleq$	Arrival time to the IR of packet i source j period k
$D^i_{j,k}$	$\triangleq$	Departure time from the IR of packet i source j period k

In this section the proof that shows instability in synchronized networks with interleaved and clock non-idealitites is presented, as explained in [20, Proposition 10].

Assume that the interleaved regulator is non-adapted: $\forall(j,p), \sigma_{f_{j,p}}^{\mathcal{H}_{IR}} = \alpha_{f_{j,p}}^{\mathcal{H}_j}$. This is, the rate and burst configured in the IR do not take into account clock deviations between devices, which is the normal configuration assuming perfect clocks. Finally, assume that the clocks $\mathcal{H}_{IR}, \mathcal{H}_{FIFO}$ and each of the $\mathcal{H}_j$ are synchronized complying with the bounded parameters stated before. Also, assume that the service curve of the FIFO system is taken as infinitive, this is, the output rate of the FIFO queue is infinitive. The proof states that for any n, ρ, η, Δ with $n \geq 3, \rho > 1, \eta \geq 0$ and $\Delta > 0$, exist an adversarial clock for $\mathcal{H}_{IR}, \mathcal{H}_{FIFO}$ and $\{\mathcal{H}_j\}_j$, and an adversarial traffic generation of the upstream flows, such that the flows have unbounded latency within the IR, when observed with any clock of the network.

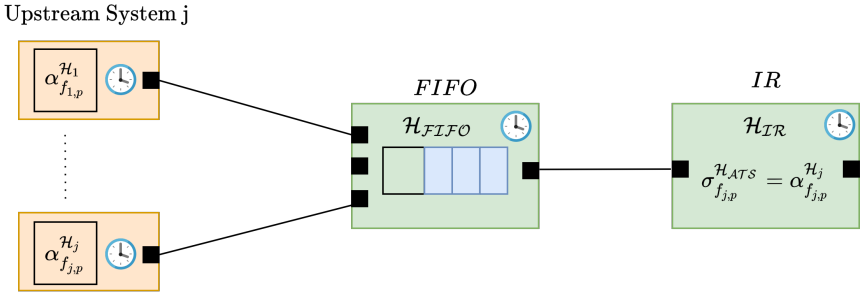


Figure 2.7: Proof scenario proposed in [20, Proposition 10]

2.3.2.1 Numerical Values for the proof

In this section the numerical proof is explained in order to validate the theory state in the section before. An adversarial clock model is proposed, meeting the constrains described before for clock models, as well as the adversarial traffic that leads to the system instability. Taking into account these requirements, the adversarial clock of figure 2.8 is proposed:

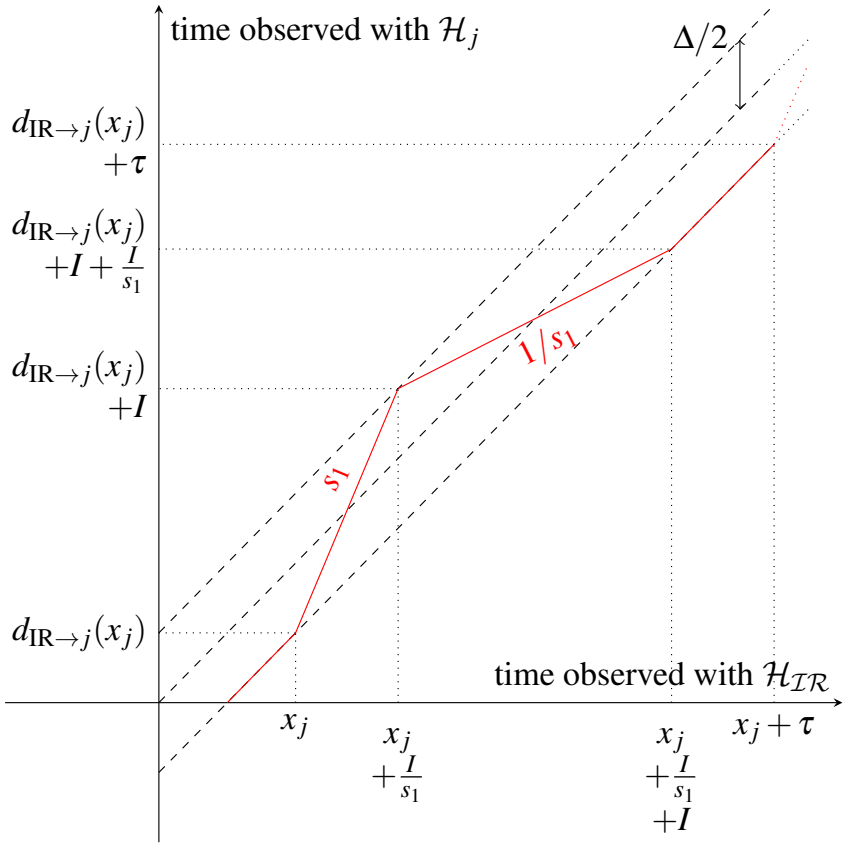


Figure 2.8: Adversarial clock model proposed in [20, Appendix J]

Figure 2.8 shows the shape of the relative time-function between the clock running in the interleaved regulator and the source. The $\mathcal{H}_{IR}$ is taken as the true time, $\mathcal{H}_{IR} = \mathcal{H}_{TAI}$, the international atomic time. Also, we consider that $\mathcal{H}_{IR} = \mathcal{H}_{FIFO}$ and the service curve of the FIFO system is infinite, this is the delay of the flows through the FIFO system is 0 when observed with $\mathcal{H}_{IR}$. The relative time-function is periodic with period τ.

The parameters for the clock model are taken as follows:

- The slope ρ is defined such as the value meets the constraints of the equation 2.1, in our case we select a s_1 value, such as $s_1 \leq min(1.5, \sqrt{\rho})$.
- A time-error bound that meets the constrains of equation 2.2 is selected, such as $d_{IR\to j}(t) - t \leq \Delta/2$.
- The interval time is defined as $I = \frac{\Delta s_1}{s_1 - 1}$.

- The period τ is defined as $\tau = nI/s_1 + n\varepsilon$.
- The initial value x_j is defined as $x_j = x1 + (j-1)I/s_1 + (j-1)\varepsilon \; \forall j = 1...n$.
- The epsilon value is defined as $0 < \varepsilon < I(1 - \frac{1}{s_1})$.

The adversarial traffic is described in figure 2.9. Each source j is configure to send a packet when the local clock of the source reaches $d_j(x_j) + k\tau$ and $d_j(x_j) + k\tau + I$. The traffic is periodic with period τ.

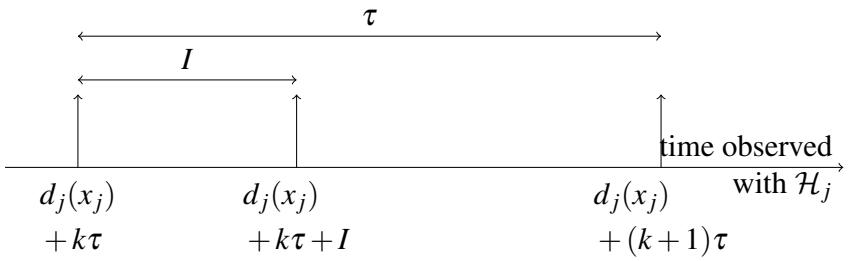


Figure 2.9: Generation of packets observed with $\mathcal{H}_j$ [20, Appendix J]

When observed with the $\mathcal{H}_{IR}$ clock, the interval time is reduced to I/s_1.

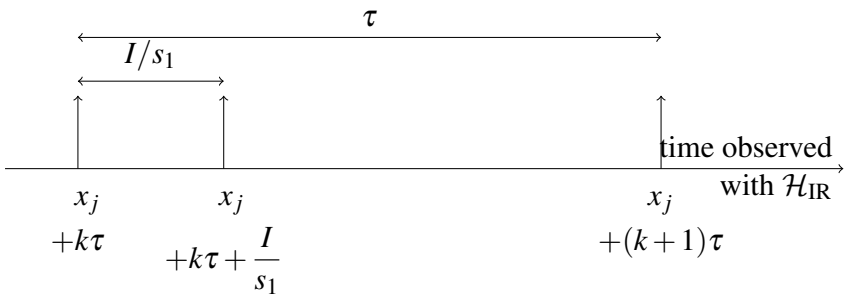


Figure 2.10: Generation of packets observed with $\mathcal{H}_{IR}$ [20, Appendix J]

The x_j value denotes the time, measure with $\mathcal{H}_{IR}$, at which sources send the first packet. Because $x_j = x1 + (j-1)I/s_1 + (j-1)\varepsilon \; \forall j = 1...n$, the interval between $A^2_{j,k}$, which is the second packet of source j in period k, and $A^1_{j+1,k}$, which is the first packet of source $j+1$ in period k, is equal to ε, when observed with $\mathcal{H}_{IR}$, as depicted in the figure 2.11.

Each source send packets within its constrains of burst and rate, measuring with the local clock an interval time equal to I. However, that same interval of time when measured with the IR clock is equal to I/s_1, because $\mathcal{H}_{IR}$ runs slower, as shown in the adversarial time model 2.8. When $A^2_{j,k}$ arrives to the IR, the regulator will introduce a delay because the interval time measured by its clock is lower than I and contradicts the regulation, as shown in figure 2.11. $A^1_{j+1,k}$ arrives to the IR before $A^2_{j,k}$ is sent, because the ε value is below the delay introduces by the IR for $A^2_{j,k}$. Since it only exists one FIFO queue for all the flows, the IR only looks at the head of the line packet, this is, the IR just observe the packet which is at the top of the queue. Thus, $A^1_{j+1,k}$ will

be delayed a value equal to $I(1-\frac{1}{s_1})-\varepsilon$, even though it doesn't contradict it's own regulation. If instead of a IR we place a per-flow regulator, we wouldn't have this problem and $A^1_{j+1,k}$ would be immediately released as soon as it arrives.

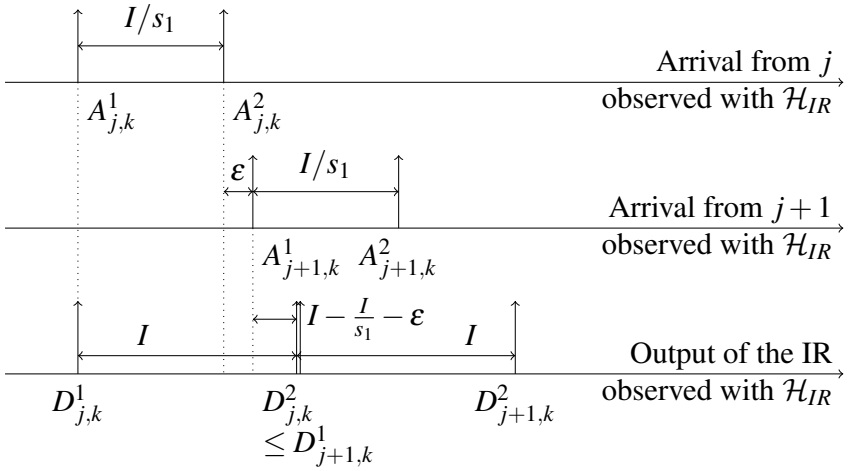


Figure 2.11: Released of packets observed with $\mathcal{H}_{IR}$ [20, Appendix J]

D denotes the release time of each packet from the IR. As we can see $A^1_{j+1,k}$ gets block by $A^2_{j,k}$ and it's release at a time $D^2_{j,k} \leq D^1_{j+1,k}$. Introducing an extra delay that will also affect to $A^2_{j+1,k}$. If the results are extended to $n \geq 3$ and $\forall k \in \mathbb{N}$, the study concludes that the introduce delay by the IR for $D^1_{j,k} - A^1_{j,k}$ in any period k is equal to:

$$D^1_{j,k} - A^1_{j,k} \geq (k-1)n\left(I\left(1-\frac{1}{s_1}\right)-\varepsilon\right) \tag{2.3}$$

As we can see, the delay introduce by the IR is completely unbounded and in each period increases a value equal to $nI(1-\frac{1}{s_1})$, leading to system instability.

In the present project we provide with a simulation proof the scenario presented in this section. Also, we propose and extension considering different service curves for the FIFO system, approaching to a more realistic scenario.

2.4 Ns-3 network simulator

In the present section I will explain the main characteristics of the network simulator ns-3 used for the project. Ns-3 is a discrete event simulator (DES) [23] written in C++ . It's a widely used open source simulator that was developed as an evolution of ns-2, due to its lack of scalability.

In this section some of the key abstractions used in ns-3 are explained. Moreover, a brief explanation about simulator execution is detailed in section 2.4.2 that will be used the following sections.

2.4.1 Key abstractions overview

Node

Node is the generic name used for a device that connects to a network. Basically, it is the basic computing device of the simulator. Functionalities, such as applications, stack protocols and network devices with their drivers can be added to the node to make useful simulations. In terms of an internet network, a node resembles to a host, end systems, router, switch or any other network element that is connected to the internet network. This abstraction is represented by the class Node.

Application

In general terms, computer software is divided in two broad classes; system software and applications. The former takes care of organizing the computer resources, such as the memory, the processor, the network, etc. The later uses these resources to execute user programs that achieve some goals. In ns-3 there is no concept of operating system, nevertheless, it can execute applications. Ns-3 applications are pieces of software that run in the nodes to perform "real word" simulations.

Channel

As in the real-world computers are connected by channels, in ns-3, nodes are also connected by the abstraction called channel. The channel class provides methods for managing communication sub-network devices. It allows to develop simple Ethernet channels to more highly sophisticated wireless channels. Examples include channels as CSMA, Point-To-Point or WIFI.

Net Device

In real networks, the connection between the hosts and the channels is done through network interface cards (NICs) and software drivers. In ns-3, this connection is done using Net Devices. Net Device abstractions cover both the software driver and the simulated hardware. Net Devices are installed in the nodes in order to enable communications with other nodes. As in the real world a node can connect to other nodes through more than one device, in ns-3 a node can have different Net Devices. For example, a Switch will have one Net Device for each specific port.

All the functionalities that ns-3 provides, are divided in different modules. A module is a set of classes that aim to provide an specific functionality, in such a manner that the user only needs to include the module in the simulation to simulate the functionalities desired. A module for example is the *CSMA* module, which includes all the classes (e.g. *Channel*, *NetDevice*) developed related *CSMA* devices. In this project two modules are created; *LocalClock* module and *ATS* module.

2.4.2 Discrete Event Simulator (DES)

Ns-3 implements by default a DES. Theoretically, the simulator tracks all the events that are scheduled by the system and execute them in their corresponding time. Events are kept sorted in time in a list of events called scheduler. Events are dequeued from the scheduler at the proper time of execution, as figure 2.12 shows. Once an event is executed, the simulator jumps directly to the next event to be executed, moving the simulation time from the last event executing time to the new event time. If for example one event is scheduled at time 50 and next event is scheduled at time 100, once the event at time 50 is executed, the simulator will jump from time 50 to time 100. All the time references in the simulation follow a unique time base which is the simulation time. This means that all the events in the simulation are scheduled following the simulation clock.

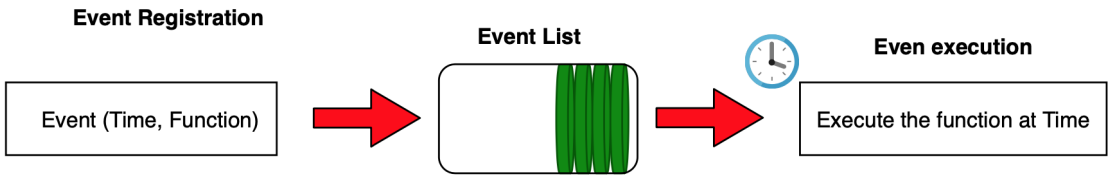


Figure 2.12: Scheduler

NS-3 system uses few things in the core of the simulator. Among them we can count;

- A simulator class, which accessed the scheduled event and executes it. It extends from the simulator interface.
- A scheduler, which inserts events in a queue sorted by their execution time.
- A notion of time, which is used to schedule events.
- An event, which is simply an action, i.e. send a packet.

Simulator Class provides a public interface for the system to schedule events. It provides 3 functions, *Schedule()*, *ScheduleWithContext()* and *ScheduleNow()* that allow to insert events in the scheduler and execute them in their corresponding time. The simulator will run until no more events are left in the scheduler or the simulator is stopped.

• *ScheduleNow()* function allows to execute events without introducing a delay, so events are

executed at the present simulator time.

• In some cases, there is a need to introduce a delay between the time that the event wants to be executed and the current simulator time. For example, an application could schedule a send packet event that it is meant to be executed in 5 seconds. To deal with this, the simulator class introduces two functions; *Schedule()* and *ScheduleWithContext()*, allowing to schedule events to be executed at some point in the future.

Occasionally, as in the debugging framework, there is a need to track which node is executing an event. This is basically the main difference between both functions. Schedule with context, allows to specify which context (node-id) is associated with the event that wants to be executed. To associate an event with a context, *Schedule()* and *ScheduleNow()* functions reuse the context of the last executed event. However, this performance is undesirable when the expected context of the reception event is that of the receiving node. For example, when there is packet transmission between nodes, a function ”receive packet” has to be scheduled in the receiving node. However, this function is schedule by the sending node, but is intended to the receiving node, so the context must change. And this is basically what *ScheduleWithContext()* provides, associate the node-id of the receiving node with the receive event.

Chapter 3

System Description

In this chapter the main design decisions to accomplish the simulation in ns-3 are explained. The chapter follows the natural steps taken throughout the project. First, section 3.1 explains how local clocks have been implemented in ns-3. This first section represents the first goal of the project, be able to simulate local clocks in ns-3. After the correct validation of this module, the implementation of ATS in ns-3 was developed, as explain in section 3.2. Hereinafter, interleaved regulators will be called Asynchronous Traffic Shaping (ATS) because is the name selected in the ongoing draft [4] of TSN and the implementation in ns-3 is based on this document. Finally, in section 3.3, as the last step, the final simulation scenario is presented .

3.1 Local clocks in ns-3

This section explains the main decision taken in the design, in order to implement the concept of clocks in the ns-3 simulator. The design has played a prominent role throughout the project. As an important goal, apart from the proof simulation, the designing of a clear and open interface for clock modelling has been established, decoupling completely the design of this module from the other modules created in order to perform the final simulation. This implies that any future researcher could create its own clock model and introduce it in ns-3 through the interface created in this project. Therefore, this also implies that the interface design has to coexist with already exiting modules of ns-3. This is not a trivial issue, because a change in the core of the simulator is proposed, without changing the exiting code of ns-3.

The idea of different time notions in ns-3 has been prowling for a long time. As a part of a work in [29], a first approach was developed in order to implement a clock per node. In this first approach, the node base class is rewritten to behave as a scheduler, in such a way, that each node has its own simulator implementation. Conceptually, this implementation makes more sense because it approaches more to the reality, where all the nodes in the network have their own scheduler of events. This would probably have led to an efficient implementation but it requires to change all the calls to the simulator by all the exiting ns-3 models. It represents a huge amount of work in rewriting many ns-3 modules, in addition to changing the Node class itself. We believe that the notion of local clocks should be enabled without changing any of the

already existing ns-3 modules.

Another project that introduced the concept of local clocks was presented in [30]. According to their paper, the authors focused on simulating IEEE 1588 synchronized networks, with very little flexibility on the clock models or applications that benefit from the local-time mechanism.

In the present project, we rely on both the ideas and the limitations of [29] and [30] and we propose an open interface that:

- allows to introduce independent local clocks, one per *Node*.
- comes as an independent module and does not require any change in already-existing modules.
- provides an interface for designing more complex clock behavioural models.
- provides an interface for updating the clock behavioural model during simulation time. For example, in an IEEE 1588 node, when a synchronization message is received, the synchronization function typically corrects the frequency of the node's clock. In ns-3, this can be simulated by having an IEEE 1588 *Application* that triggers an update of the clock model (with the new frequency), using the interface that we provide. This interface is the basis for analysing the interaction between local clocks and network events/mechanisms.

3.1.1 Proposed design

As the majority of network simulators, ns-3 only introduces the concept of global time. Events in the simulator are only scheduled within one timeline, which is the same for all the nodes in the network. However, for some kind of networks and simulations, there is a need of simulating a local time different from the simulation time. In general, we refer to the local time as node clock or local clock and global time as simulation time or true time.

A *clock* module design is proposed, which is composed by 2 classes and one interface, as depicted in figure 3.1.

In order to better understand the functionality of the module, three study cases are proposed:

- How to enable *clock* module and already ns-3 existing modules to coexist.
- How to implement and model local clocks in ns-3.
- How to proceed when a clock update is triggered during the simulation.

3.1.1.1 Integration of modules

The first question to answer is how to integrate modules. As explained in section 2.4.2, all the processes in ns-3 (e.g. an application), call to the same three functions to schedule events; *Schedule()*, *ScheduleWithContext()* and *ScheduleNow()*. When a call is made to schedule an event, the call is forwarded to the simulator implementation attached to each simulation.

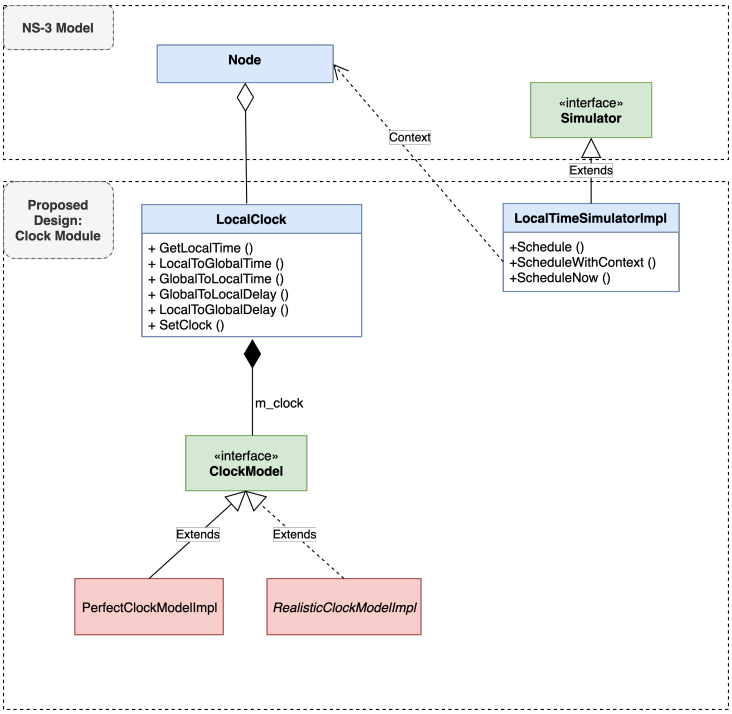


Figure 3.1: UML design

This forces all the nodes within the simulation to schedule events in the same simulator implementation. In order to be able to implement our *clock* model within the existing ns-3 models, we need to keep a unique simulator implementation to where all the existing calls must be forwarded. If we want to implement local times in ns-3, the scheduled events should be executed in a time that depends on the local clock of the node to which the event belongs. As it stands, the default simulator implementation provided by ns-3 schedules all the events without taking into account this consideration. We propose to design a new simulator implementation class, called *LocalTimeSimulatorImpl*, extending from the simulator interface, as shown in figure 3.1. *LocalTimeSimulatorImpl* allows to schedule events, taking into account different time notions. Minimal changes need to be done in order to change from the default simulator implemented in ns-3 to *LocalTimeSimulatorImpl*. In such a way that, if well configured, all the calls to schedule events that already exists in ns-3 modules will be executed by the *LocalTimeSimulatorImpl* class. The idea behind *LocalTimeSimulatorImpl* class is that events are scheduled depending on the local clock of the node to which the events belong. To this effect, we need two things, how to attach clocks to nodes and how to model clocks.

3.1.1.2 Implementation and modelling of clocks

A *LocalClock* object is created and aggregated to every Node using the ns-3 aggregation system, as figure 3.1 shows. It represents the interface between the clock model and the node and contains as a main attribute a pointer to the clock model implementation, which is the object that behaves as the real clock. The aggregation system is one of the most powerful characteristics of ns-3. It allows to overcome the weak base class problem. In our case, the class *Node* was designed without any concern about local clocks. This problem could be solved by extending

the base class to a *NodeClock* class. However, this would imply to developed a new whole class, which would be very similar to the base class in addition to change all the node objects of all the ns-3 modules. To overcome this problem, ns-3 rises the concept of aggregation. Objects with no inter-dependency between them can be aggregated, in such a way that from one object you can access the other object and vice versa. This allows to add new capabilities to base classes without modifying them.

In our case, this functionality is very useful, it allows to aggregate a *LocalClock* object to the *Node* object, in such a way that each node has a clock attached to it. Each time an event is scheduled, *LocalTimeSimulatorImpl* is able to retrieve the *Node* object to which the event belongs. In this manner, *LocalTimeSimulatorImpl* class accesses the *LocalClock* object through the node, while keeping the *Node* class untouched. The communication between *LocalTimeSimulatorImpl* and node is only possible using the context, which is the node-id to which the event belongs. Figure 3.2 explains the main steps taken by the *LocalTimeSimulatorImpl* when scheduling events.

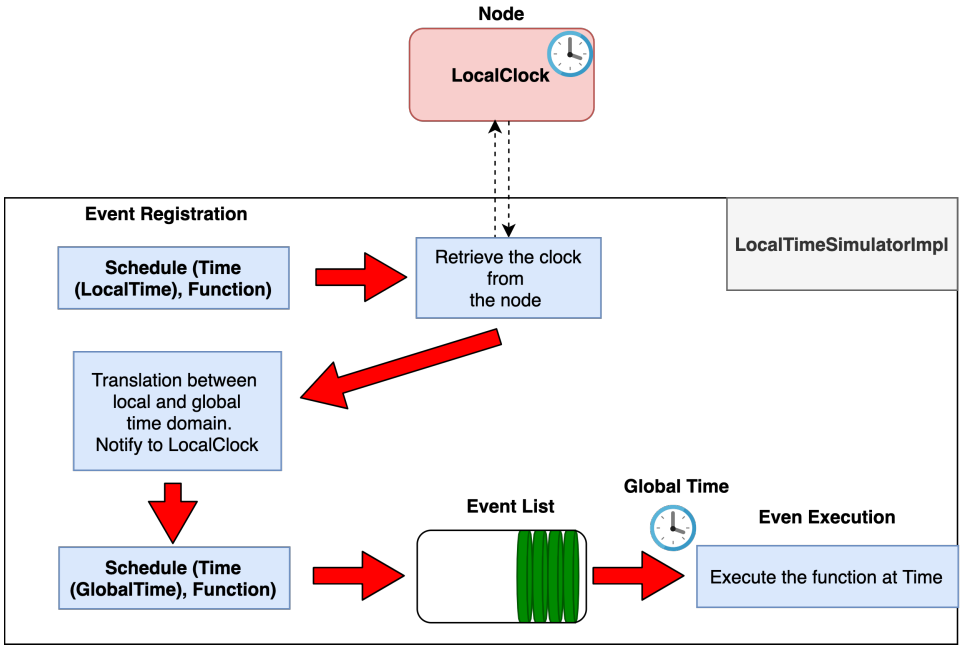


Figure 3.2: Simulator steps when *Schedule()* or *ScheduleNow()* are called

In order to introduce the notion of local time in ns-3, the clock model must provide a mapping function between the local time and the simulator time. We introduce in ns-3 the concept of time-relative functions, as explained in 2.3.1. The relative time-function maps two different time domains, capturing clock non-idealities between both clocks. This appears to be a good way to model clocks in ns-3. As stated in [10, Annex C] clock deviations are computed based on a pair of oscillators. The minimum set of functions that any clock model implementation should have, is defined by *ClockModel* interface. Any clock model implementation must extend from this interface, as shown in figure 3.1.

In order to provide with this mapping capabilities to *LocalTimeSimulatorImpl*, *Schedule()* and

ScheduleNow() functions are modified. The only difference between both functions is that when *ScheduleNow()* is called, the call is redirected to *Schedule()* function with a time equal to 0. The main steps of *LocalTimeSimulatorImpl* when *Schedule()* and *ScheduleNow()* are called, are gathered as follows:

- Event registration: An event, which is a function and the time at which that function has to be executed, is registered in the simulator. The time is the delay, measured by the local clock, between the registration time and the execution time of the function.
- Clock retrieval: After the registration, the clock associated to the event must be retrieved. For that, the context of the event is used. The context allows to associated the event with the node that registered the event. Once the *node* object is retrieved, the *LocalClock* object is obtained using the aggregation system.
- Time conversion: Once the *LocalClock* object is obtained, the methods that translated between time domains can be accessed, allowing to translate between the event local time, measured by the local clock, and the global time. Also, it notifies to the *LocalClock* object that the event has been finally scheduled. The explanation of this last step will be understood in the following section.
- Schedule event: Finally, the event is inserted in the scheduler for subsequently execution.

Most of the *ScheduleWithContext()* calls in today's ns-3 modules are related to a packet transmission within a channel. The delay is normally the channel propagation time, which does not depend on the node clock. Because of that reason, when calling *ScheduleWithContext()* no mapping between times happens.

Consider the following example shown in figure 3.3, with one node. *Node1* runs a local clock which is slower than the global clock. It schedules events E_1 and E_2 at *init time*. The events are meant to be executed at some point in the future. The delay between the *Init time* and execution time is measured by the local clock of the node. To be introduced in the simulator, the delay needs to be translated into a global delay. Because *node1* clock runs slower, the final delay introduced in global time is bigger than the local delay. The event is finally executed at a more distant time than the one measured by the local clock. All the events of the simulation are scheduled in the same timeline, which is the global time. However, events can be scheduled shifted in time depending on the deviations between local and global clocks.

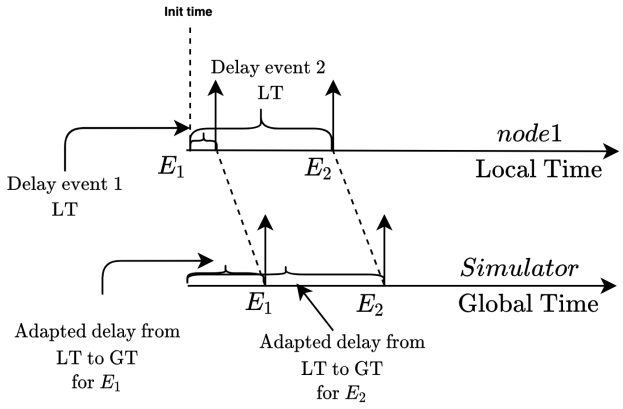


Figure 3.3: Example of event scheduling

3.1.1.3 Clock update

In this case, we suppose that a clock update is triggered during the simulation. The clock update can be triggered due to the reception of a clock synchronization message that force to change the clock model implemented in the node. Nodes within the simulation may schedule lots of events to be executed in the future. If the node clock changes, all the events that have been already scheduled should change their global time at which are supposed to be executed. The mapping between local and global time is no longer valid and a new time has to be recomputed using the new clock. This problem arises a number of questions gathered as follows:

- How to control all the scheduled events that need to be rescheduled by a node. The scheduler in the simulator is the same for all the nodes.
- How to recompute the times.
- How to reschedule the events.

In order to cope with the different issues that arise with the rescheduling functions, all the logic is implemented in the *LocalClock* object of the node, decoupling all the rescheduling algorithm from the simulator and reducing the complexity of the system. The node maintains an updated list of the events that have been scheduled by it. When the local clock is updated, the *LocalClock* object of the node is in charge of triggering all the actions. The different steps taken by the *LocalClock* object are as shown in figure 3.4 and gathered as follows:

- Events retrieval: When a clock update is triggered in the node, the *LocalClock* object attach to the node retrieves the events that need to be updated. This list of events has been filled by the *LocalTimeSimulatorImpl* object when scheduling events.
- Time re-computation and event registration: In this step the *LocalClock* computes the remaining time in local time and registers a new event with the new time updated. Also, it removes from the simulator the events that do not want to be executed because their time is not valid anymore.

- Schedule the event in the simulator: Now the simulator will retrieve the new clock from the node and will perform the same operations explained in the last section.

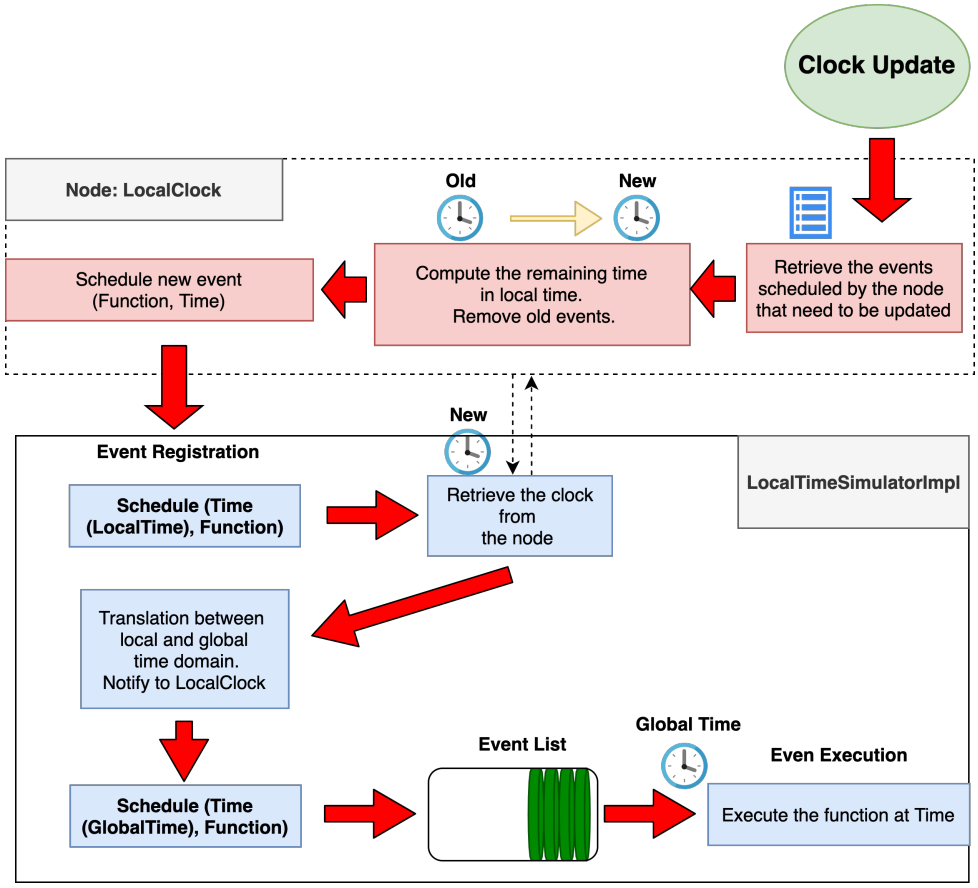


Figure 3.4: Clock Update steps

Following the same example as before, a clock update is triggered before E_2 is executed. Therefore, E_2 needs to be updated by the node as shown in figure 3.5. In this case, the new clock runs even slower than the old clock, which means that the new event $E_2^{'}$ has to be delayed even more. A new event $E_2^{'}$ is scheduled and E_2 is removed.

Appendix A shows the validation example used for the module. A basic configuration of a clock model is presented, as well as the process to attach the clock to the node. The main results are also discussed. Also, the performance evaluation and the limitations of the module are discussed in appendix B. The code of the module can be found in the repository [7] in V1-Clock brunch, as well as a detailed document about the module in [19], used for the merge request to the ns-3 project. The merge request discussion can be followed in [28].

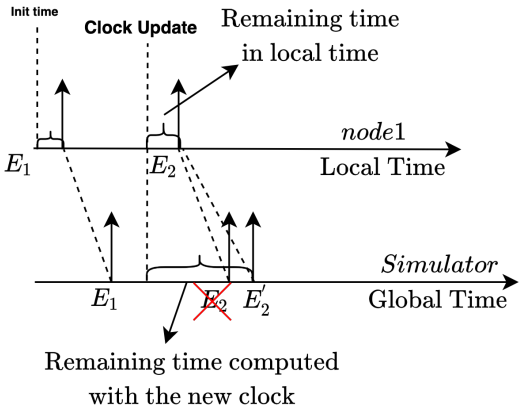


Figure 3.5: Example of clock updating

3.2 ATS in ns-3

In this section, the main underlying design concepts behind the ATS module are explained. ATS implementation in a bridge, as explained in the draft [4], has lots of features, such as stream gating, policing, etc. Some of the features explained in the ongoing draft of ATS are not implemented in the present project, we focus the deployment on the main characteristics of the shaping functionality. Nevertheless, the ATS design introduced in ns-3 is completely modular and flexible, which means that anyone could create new features for the ATS bridge and append them to the code created in this project, without the need of rewriting it.

ATS theoretical performance has been studied widely. However, few papers are related with the simulation performance. For instance, the paper [27] studies the design of relatively accurate models for ATS, measuring the performance in a simulation scenario. A software modeler is used for the simulation, nevertheless, clocks within the simulator are assumed to be perfect.

In this section, an ATS design in ns-3 is proposed, which:

- allows to reshape traffic at the output port of any network element (e.g. bridge) in ns-3.
- follows closely the implementation patterns dictated in 802.1Qcr for TSN networks [4].
- is modular and provides flexibility to extend the system.

3.2.1 Traffic Control Layer

In order to implement the shaping functionality in the output ports of network devices, the already existing Traffic Control Layer (TCL) module of ns-3 is used. This module, which is similar to the Traffic Control Layer of Linux [18], supports the operations needed to provide a quality of service (QoS), including policing, scheduling, shaping, etc. It provides a set of queues and mechanisms by which packets can be treated, applying a QoS before being transmitted trough the output port. The TCL module of ns-3 already provides all the interconnection points

between the output port and the traffic control object, allowing to enqueue the outgoing packets in a queue discipline and preform multiple actions on them, as seen in figure 3.6.

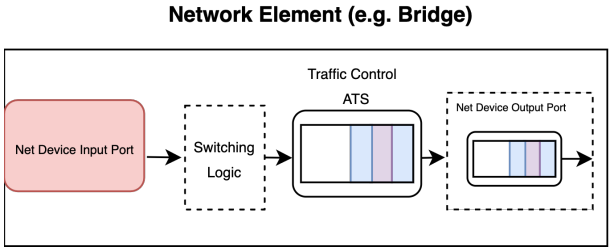


Figure 3.6: Traffic Control Layer structure in a bridge

The traffic control layer is divided in three fundamental groups:

- Queue Disc: is the scheduler. Every output needs a scheduler of any kind and the simplest scheduler is a FIFO queue.
- Classes: can contain either multiple children classes or a single queue disc.
- Filters: allow to classify the traffic within the classes and apply policies on the outgoing traffic.

Traffic control layer allows complex structures with more than one queue disc or class, building tree structures, as seen in figure 3.7. A tree structure can be built with a root queue disc and an undefined number of queue classes and filters attached to it. This functionality allows to build hierarchical structures where different schedulers can be placed in different branches of the structure.

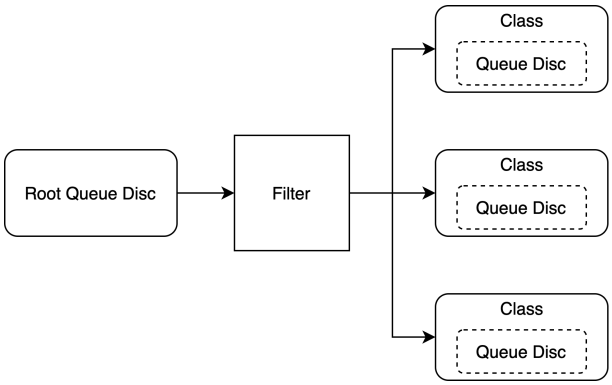


Figure 3.7: Traffic Control Layer structure

The TCL structure adapts well with the concept of class-based queuing adopted in time-sensitive networks with ATS. All outgoing frames of a bridge are classified into several classes and shaped by an ATS, as shown in figure 3.8. Remark that a traffic class can have different flows and each flow is treated with a different ATS scheduler. Classes can be also further classified

into more classes, as proposed by the traffic control layer, building more complex structures. In figure 3.8 the root queue disc is represented by the scheduler. In the TCL structure, all outgoing frames enter first the root queue disc discipline. This root queue disc discipline does not apply any operation in the packet, it just forwards the packet for further classification by the filter. Finally, the frame is enqueued in one of the queue disc of the classes. However, when dequeuing frames, it's the root queue disc who selects a class from where the frame must be dequeued. The root queue disc and the scheduler could be, for example, a fixed priority queue discipline [14] or a Round Robin scheduler [16]. However, it's not the purpose of this project to implement these schedulers, we only focus on the design of the ATS implementation but we open up the possibility to build more complex structures.

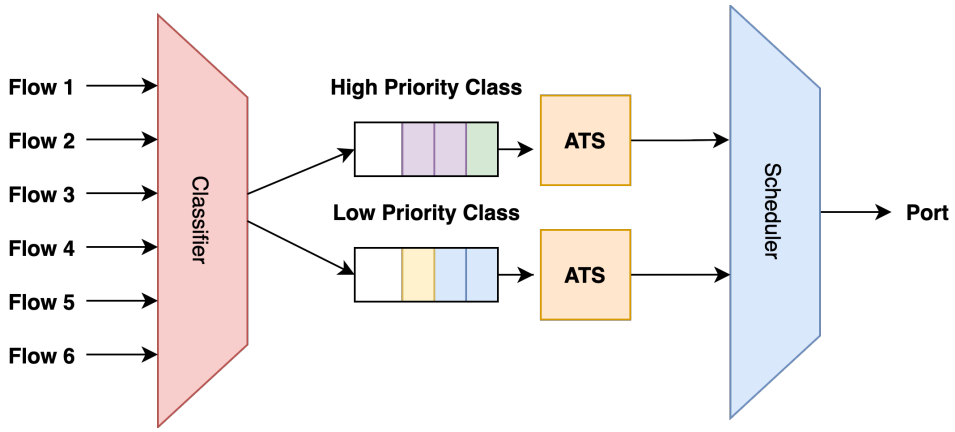


Figure 3.8: Per class-base queuing with ATS

In the present project, we make use of the traffic control layer of ns-3. It provides the modularity that is needed, as well as the simplicity to fit different queuing discipline in a well-designed module. The base classes for the queue disc and filter are already designed and we just need to extend from those base classes to build our models. However, it forces some constrains that limit the design, as we discuss in the next section. Appendix C presents the UML design carried out for this module.

3.2.2 ATS architecture in ns-3

Figure 3.9 shows the main components of the ATS module in ns-3. It represents the structure placed in each of the output ports of a bridge device. Following the structure of the traffic control layer, an ATS Transmission Queueing discipline is placed as the root queue disc. This queuing discipline has as a main scheduler a FIFO queue and it contains all the packets that are ready to be transmitted by the output port, this is, the packets that have been selected by the ATS transmission algorithm. Attached to this root queue disc, we find the classifier, also called the stream filtering by the 802.1Qcr draft nomenclature. The classifier will classify the different flows into the different ATS schedulers. Mention that for the purpose of this project, no stream gates or stream filters are provided, as shown in figure 2.5. However, this could be easily solved by adding to each class a stream gate or stream filter module. As it can be seen in figure 3.9, this policing requirements are shown but are not implemented in the module, thus, flows are

classified directly into the ATS schedulers. This ATS schedulers represent the leaf's of the TCL structure and perform the shaping functionalities of the ATS.

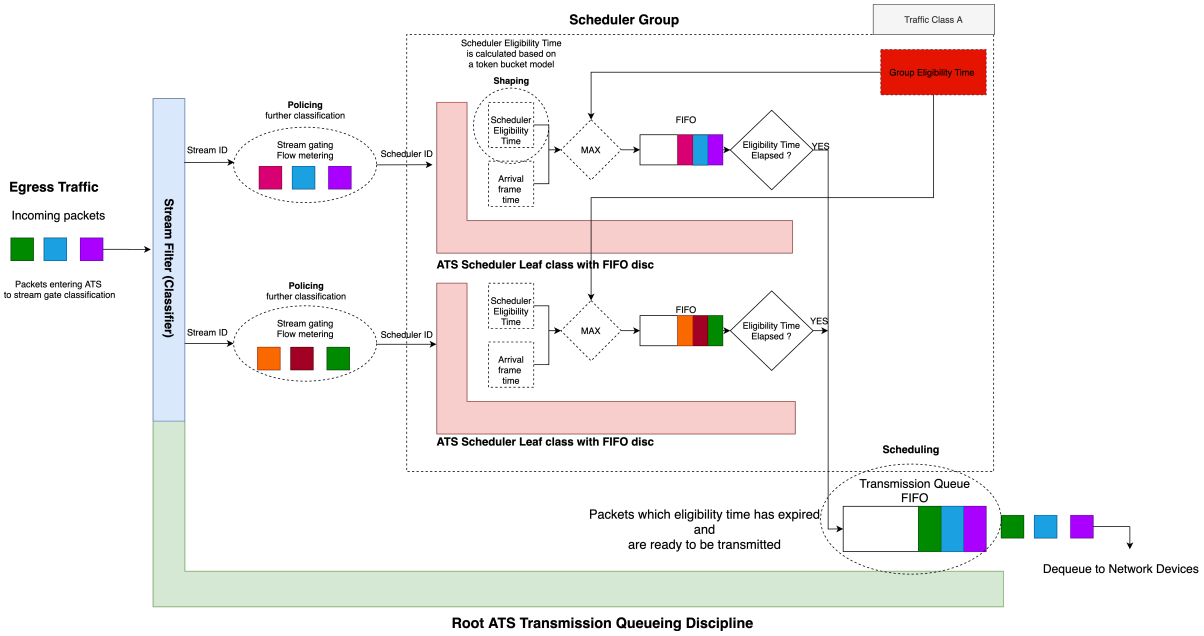


Figure 3.9: ATS design in ns-3

3.2.2.1 Forwarding process of the ATS structure

When frames arrive to the ATS structure, the classifier classifies them between the different ATS schedulers. The classification can be done depending on a wide variety of parameters, as explained in [4, Section 8.6.5.1]. However, in the case of this project, the flow classification is done based on the source MAC and destination MAC of the frames. Nevertheless, this can be easily changed by redesigning the filter. The ATS schedulers are the part of the ATS structure that perform the shaping functionality. The eligibility time is calculated based on the algorithm explained in the following section 3.2.2.2. For the time being, we only need to know that the algorithm calculates an eligibility time for each frame that enters the ATS structure. The eligibility time is the time at which the frame should be sent to the output port of the device. Taking advantage of the fact that we are working on software, the frame is directly enqueued in a FIFO system that belongs to the ATS scheduler. In order to dequeue the frame, an event is scheduled in the simulator at the eligibility time calculated for the frame. In such a way, that when the event is executed, the frame is dequeued from the ATS scheduler and enqueued in the ATS transmission queuing discipline. Because the eligibility times are calculated in a non-decreasing order, we still maintain the order when dequeuing frames. This is basically the implementation of the ATS transmission algorithm. Recall that the ATS transmission algorithm determines the frames eligible to be transmitted, and in this case, this mechanism is applied when the event scheduled at the eligibility time of the frame is executed.

We can appreciate that this model contradicts the very first statement of ATS or interleaved regulators (IR), where one FIFO queue is placed per input port, output port and traffic class. In

proposed the model, there exists one FIFO queue per ATS scheduler, which truly appears to be the behavioural model of per-flow regulators. This limitation is due to the base classes already provided by the traffic control layer of ns-3. The base classes have been designed forcing to implement a queuing system in each of the leafs of the structure. However, we force the group eligibility time in all the ATS schedulers that belong to the same class. This allows to assign eligibility times in a non-decreasing order for ATS schedulers that belong to the same group, treating frames of the same class in a FIFO manner. Because we are in a simulator, we do not care about placing one FIFO queue per flow, nevertheless, in reality, this implies to place much more hardware resources, being this one of the drawbacks that exits for per-flow regulators.

3.2.2.2 ATS scheduler algorithm

All the ATS schedulers have their own shaping parameters *CommittedInformationRate* and *CommittedBurstSize*, which are used to shape the flows. The ATS scheduler algorithm is a modified version of the well-known Token Bucket Filter (TBF) of Linux [17]. Each time a frame arrives, the frame is processed with the algorithm explained in algorithm 1, as explained in [4].

Algorithm 1 ATS scheduler algorithm

1: **procedure** PROCESSFRAME($frame$)
2: $lengthRecoveryDuration = \frac{length(frame)}{CommittedInformationRate}$
3: $emptyToFullDuration = \frac{CommittedBurstSize}{CommittedInformationRate}$
4: $schedulerEligibilityTime = \frac{BucketEmptyTime}{lengthRecoveryDuration}$
5: $bucketFullTime = \frac{BucketEmptyTime}{emptyToFullDuration}$
6: $eligibilityTime = max(arrivalTime(frame), GroupElibilityTime, schedulerEligibilityTime)$
7: **if** $eligibilityTime < arrivalTime(frame) + MaxRecidenceTime/1.0e9$ **then**
8: $GroupElibilityTime = eligibilityTime$
9:
10: **if** $eligibilityTime < bucketFullTime$ **then**
11: $BucketEmptyTime = schedulerEligibilityTime$
12: **else**
13: $BucketEmptyTime = schedulerEligibilityTime + eligibilityTime - bucketFullTime$
14: **else**
15: $Discard(frame)$

The *lengthRecoveryDuration* is the duration that is required to accumulate a number of tokens equal to the frame length in seconds.
The *empyToFullDuration* denotes the time that it takes to fill the token bucket at the committed burst and rate.

The *bucketEmptyTime* is a variable that contains the most recent instant of time at which the token bucket of the ATS scheduler is empty. It is initialized with a time earlier than the division between CBS and CBR in the past, which actually is the *emptyToFullDuration*.
The *SchedulerEligibilityTime* is the time at which the number of tokens in the bucket is equal or bigger than the frame size.
The *BucketFullTime* denotes the time at which the bucket is full of tokens and equal to *CommittedBurstSize*.
The final *eligibilityTime* is calculated as the maximum value between the arrival time of the frame, the *GroupEligibilityTime* and the *SchedulerEligibilityTime*.

If a frame arrives with a time bigger than the times calculated by the *SchedulerEligibilityTime* and *GroupEligibilityTime*, the frame will be immediately released. The *GroupEligibilityTime* is the most recent *eligibilityTime* from the previous frame processed by any other shaper in the same group, assigning times in a non-decreasing order for ATS schedulers that belong to the same group, being the latter the main difference between ATS or Interleaved regulators and per-flow regulators. In order to perform as a per-flow regulator, the *GroupElibilityTime* can be deactivated. Once the eligibility time is calculated, the scheduler checks if the value is below a maximum time delimited by the *MaxResidenceTime*. If the value is below, the *GroupElibilityTime* is updated. In order to calculate the *BucketEmptyTime*, the algorithm checks if the eligibility time is below the time that it takes to fill the bucket of tokens. If not, the next frame would be directly affected because the *BucketEmptyTime* would have a bigger value than the *schedulerElibilityTime*.

As it can be seen, the algorithm enforces a strong relationship between the previous frame size and the arrival time of the actual frame. A frame can be delayed because of regulation constrains tuned on the ATS scheduler, or because of the group eligibility time. On contrast, the frame is directly sent, if the arrival time of the frame is bigger than the *SchedulerEligibilityTime* and the *GroupEligibilityTime*, allowing to apply the shaping for free property.

In order to validate the well performance of the ATS module, the carried out tests are presented in appendix D. The code related to the ATS module can be found in [7] in the ATS branch.

3.3 Simulation implementation

Following the scenario model proposed for the the theoretical proof presented in chapter 2, the scenario shown in figure 3.10 is proposed. The scenario model includes the modules explained in this chapter. On the one hand, the clock module is used to attach a clock model to each node of the scenario. On the other hand, the ATS module is attached to the output port of the bridge implementing the shaping functionalities. We divide the scenario model in three different parts; the sources and the consumer, the bridge with the FIFO system and finally the ATS bridge.

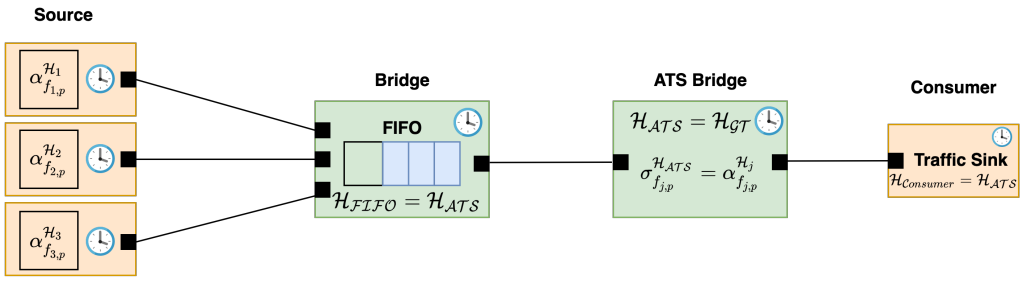


Figure 3.10: Scenario model

Sources and consumer

The consumer is the traffic sink that receives all the traffic from the sources and it's placed only with tracing purposes. From the point of view of the classifier of the ATS system, all the flows created by each source belong to the same traffic class, sharing the same group eligibility time. In order to create the specific traffic pattern described in chapter 2 section 2.3.2, an application is designed in ns-3. The designed application can be tuned with the parameters that characterize the traffic, such as the rate and burst, the time at which the application starts sending packets and the period. Moreover, each source has its own implementation of a local clock. The clock model implemented in each node has the same relative time-function shown in figure 3.11. The difference between the source clocks resides in the initial value x_j configured. The time relative function maps between the global time or, in this case, the ATS time and each of the clocks $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ attached to each source. In order to implement this clock model in ns-3, the following function has been used, as proposed in [20]:

$$d_{\text{ATS}\to j} = \begin{cases} t - \Delta/2 & \text{if } t \le x_j \\ s_1(t - x_j) + x_j - \Delta/2 & \text{if } x_j < t \le x_j + I/s_1 \\ \left.\begin{array}{r} 1/s_1(t - x_j + I/s_1) \\ +I + x_j - \Delta/2 \end{array}\right\} & \text{if } x_j + \dfrac{I}{s_1} < t \le x_j + \dfrac{I}{s_1} + I \\ t - \Delta/2 & \text{if } x_j + \dfrac{I}{s_1} + I < t \le x_j + \tau \\ \tau + d_j(t - \tau) & \text{if } x_j + \tau < t \end{cases} \tag{3.1}$$

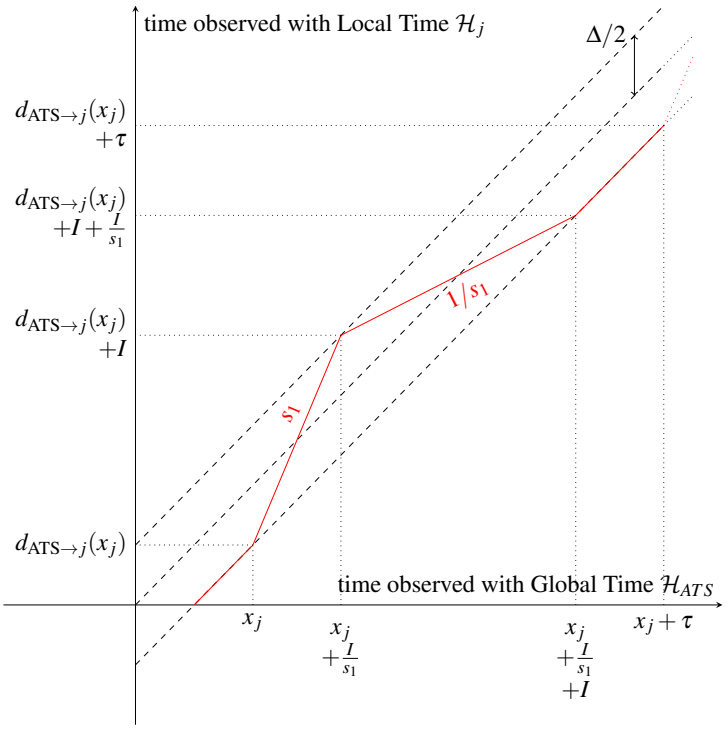


Figure 3.11: Adversarial clock, as proposed in [20]

Bridge with the FIFO system

In order to simulate the FIFO system placed after the sources, a bridge is needed in terms of the ns-3 scenario model. This bridge is a layer 2 device which will forward frames based on the MAC addresses. In the output port of the bridge we place a drop tail queue, which is a FIFO system that will drop packets when the queue starts building-up. Due to the clock framework being used, a local clock in the bridge is needed. In this sense, the local clock implemented is the same as the ATS bridge clock, which at the same time is equal to the simulator clock, this is, the clock runs perfectly synchronized with the simulator clock. In this first approach we consider that the output rate of the bridge is infinite. Packets cannot be delayed in the FIFO queue, so the only purpose of the FIFO system is to send through the same output port all the flows, in such a way that all the flows build-up in the same queue system in the ATS, sharing the same group eligibility time. Recall that in ATS or IR's, the idea is to have one queue per input port output port and traffic class.

ATS bridge

The ATS bridge is simply a switch that implements in the output port the ATS model developed in the section before. The ATS module is configured with three ATS schedulers, one for each flow. The parameters are non-adapted, $\sigma_{f_{j,p}}^{\mathcal{H}_{ATS}} = \alpha_{f_{j,p}}^{\mathcal{H}_j}$, which means that rate an burst of the ATS schedulers are equal to the rate and burst of the flows and they do not take clock deviations into account. All the ATS schedulers belong to the same ATS group, forcing the group eligibility time. As mentioned before, the clock in the ATS bridge runs perfectly synchronized with the

simulator clock.

The code related to the application and the clock model can be found in [7], in the folder *src/applications/model/adversarial-generation.cc* and *src/clock/model/adversarial-clock.cc* of the ATS branch.

Chapter 4

Simulation Results

The results of the simulation for the proof presented in Chapter 2 section 2.3 are presented in this chapter. Along with the results, the parameters tuned for the simulation are explained. Finally, an extension of the results is proposed, both mathematically and by simulation.

4.1 ATS instability

In this first simulation, the ATS instability is presented under the specific traffic and clock models. The parameters chosen for the adversarial traffic generation, the clock model and the ATS schedulers are shown in table 4.1. The values selected for this simulation do not have any specific meaning. They are values within the bounds established by the proof but do not correspond to any type of network. The ATS schedulers parameters slightly variate from the traffic model parameters, due to the packet size when processed. The traffic packets are created at the application layer, so when they are processed by the ATS scheduler, the UDP (8 bytes) +IP (20 bytes) +Ethernet (18 bytes) headers must be added.

In figure 4.1 the delay introduced for the 10 first packets of each flow through the ATS is plotted. The delay is equal to the subtraction between the released time from the ATS bridge, in other words, the eligibility time and the time at which the packet enters the ATS, the arrival time. As we can observe, the delay introduced by the ATS is completely unbounded and increases for every packet leading to the system instability.

Time model	**Source 1**	**Source 2**	**Source 3**
Δ (μs)	1	1	1
s_1	1.001	1.001	1.001
Interval I (ms)	10	10	10
ε (μs)	0.5	0.5	0.5
Period τ (ms)	29.97153	29.97153	29.97153
$x_j(ms)$	5	14.990001	24.98002
Traffic model	**Flow 1**	**Flow 2**	**Flow 3**
Peak Data Rate (KBPS)	51.2	51.2	51.2
Packet Size (bytes)	512	512	512
Traffic type	UDP	UDP	UDP
ATS scheduler	**Flow 1**	**Flow 2**	**Flow 3**
Data Rate (KBPS)	58.0	58.0	58.0
Burst Size (bytes)	558	558	558

Table 4.1: Simulation parameters

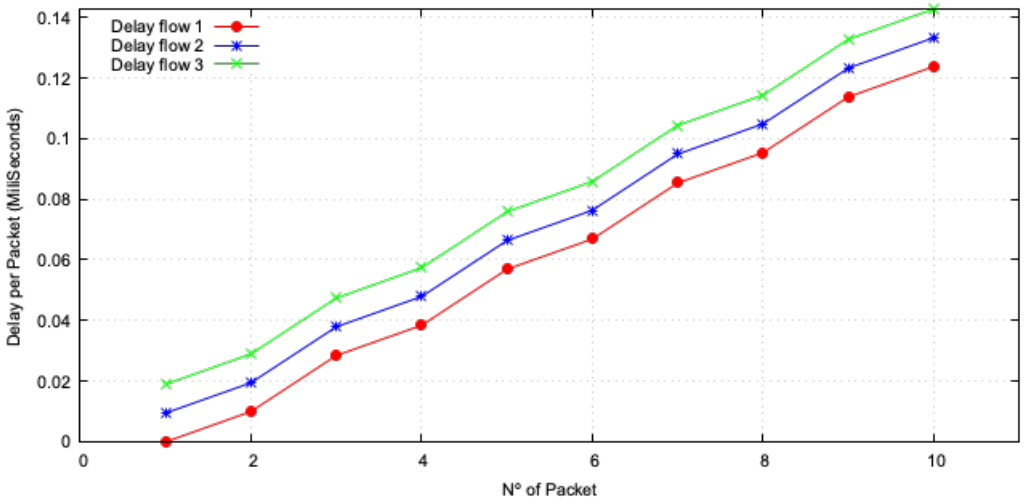


Figure 4.1: Delay per packet introduced by the ATS

At this point, we want to validate if the simulation follows the proof explained in section 2.3.1. In order to help with the evaluation of the results, the delay for the first 4 packets of the graph are represented in table 4.2, as well as a reminder of the traffic pattern in figure 4.2. Recall that each source j outputs two packets each period k with a time interval equal to I when measured by the source clock. The interval time between packets of different flows in the same period is equal to ε.

Packet N^o	Flow 1	Flow 2	Flow 3
1	0	0.00949	0.01898
2	0.00999	0.01948	0.02897
3	0.02847	0.03796	0.04745
4	0.03846	0.04795	0.05744

Table 4.2: Introduced delay by the ATS in milliseconds for the first 4 packets of each flow

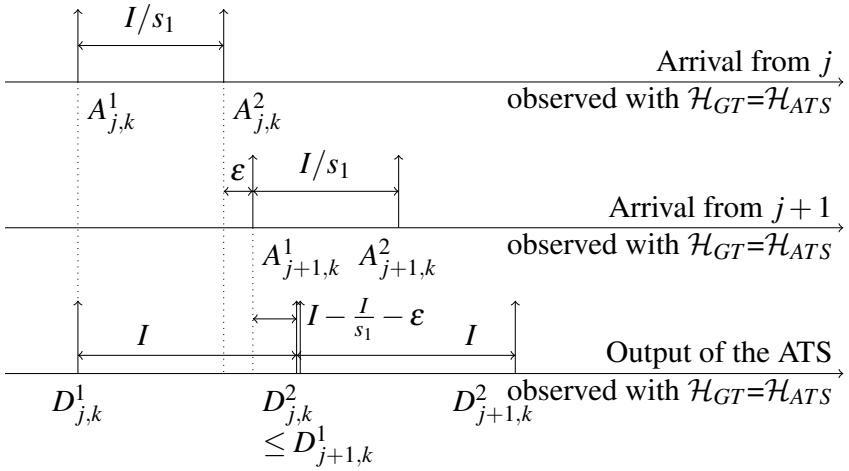


Figure 4.2: Delay introduced by the ATS when observed with $\mathcal{H}_{ATS}$, as proposed in [20]

First, we focus on the delay introduced by the ATS for packet $A^2_{j,k}$ shown in figure 4.2. No other traffic exists before $A^1_{j,k}$ arrives, therefore, $A^1_{j,k} = D^1_{j,k}$. When packet $A^2_{j,k}$ arrives to the ATS, it is too soon, because when measured by the clock of the ATS, the interval time is equal to $\frac{I}{s_1}$. For this reason, the introduced delay by the ATS is equal to $I - \frac{I}{s_1}$. This first observation, allows to confirm that the ATS is shaping the traffic. When comparing this value to the one obtained for packet $A^2_{j,k}$ in the table 4.2, a delay of 0.00999 ms is obtained, which actually is equal to the theoretical value $I - \frac{1}{s_1}$. In reality, the values do not perfectly match because the simulator has a resolution when performing operations, therefore, some of the results are truncated.

The packets between flows are sent with an ε interval between them, in such a way that the ε value is upper-bounded by $\varepsilon < I(1 - \frac{1}{s_1})$. Because $A^2_{j+1,k}$ arrives before $A^1_{j,k}$ is sent, $A^2_{j+1,k}$ is automatically delayed. This is due to the very basic property of ATS or interleaved regulators, where we only have one FIFO queue for all the aggregated flows, in such a way that the ATS only looks at the head of the line packets. In the case of our specific implementation this is forced by the group eligibility time of the ATS. As we can see in table 4.2, the delay introduced for $D^1_{j+1,k} - A^1_{j+1,k} = 0.00949$ ms is equal to $I - \frac{I}{s_1} - \varepsilon$. The value obtained proves that the group eligibility time of the ATS has been taken into consideration.

Finally, we validate the equation 2.3 that shows the instability in the ATS. This equation states

that delay suffered through the ATS by the first packet of the kth period of the first source increases linearly in each period. For this purpose the first packet of the kth period of the first source is isolated in figure 4.3:

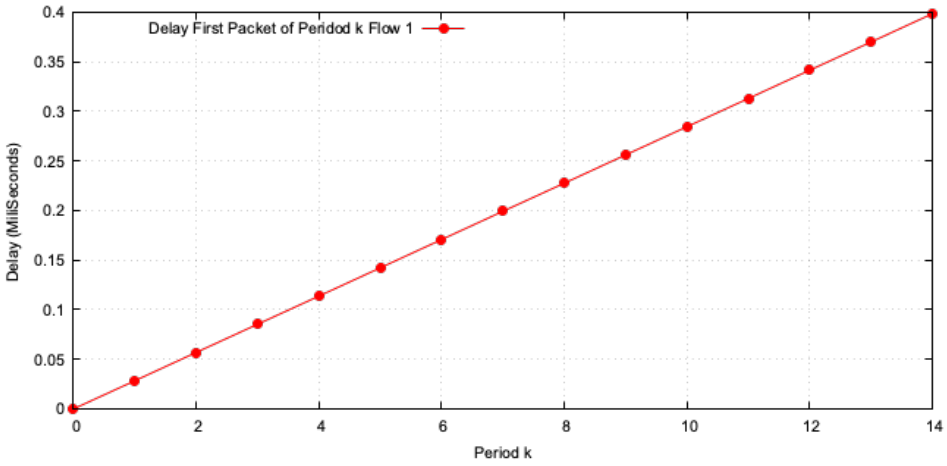


Figure 4.3: Delay of the first packet in period k for flow 1

The simulation clearly proves that the delay increases linearly. Using the data obtained from the simulation, we obtain a value of $28.47\mu s$ increase per period, which actually is the same as the one obtained using the equation 2.3 for the first period. We can state, as shown in the figure 4.3, that the delay diverges as k increases, therefore the delay though the ATS is unbounded when seen from $\mathcal{H}_{ATS}$ perspective, which proves the instability.

Complementary simulations show that the instability of the ATS does not depend on the synchronization accuracy. The delay through the ATS diverges at the same rate in "loosely" synchronized networks ($\Delta = 100ms$) and "tightly" synchronized networks ($\Delta = 1\mu s$).

4.1.1 Different ε values

Recall that the ε value, as seen in figure 4.2, is the interval time between packets of different sources in the same period. The proof states that an arbitrary value ε can be selected such that $0 < \varepsilon < I(1 - \frac{1}{s_1})$. Figure 4.4 shows the delay suffered through the ATS by the first packet of the kth period of the first source, when different ε values are selected. We show that the instability is maintained if the ε values are within the bounds provided. The closer we get to $\varepsilon = I(1 - \frac{1}{s_1})$, the slower is the rate at which the delay diverges. However, the instability maintains for values of $\varepsilon < I(1 - \frac{1}{s_1})$.

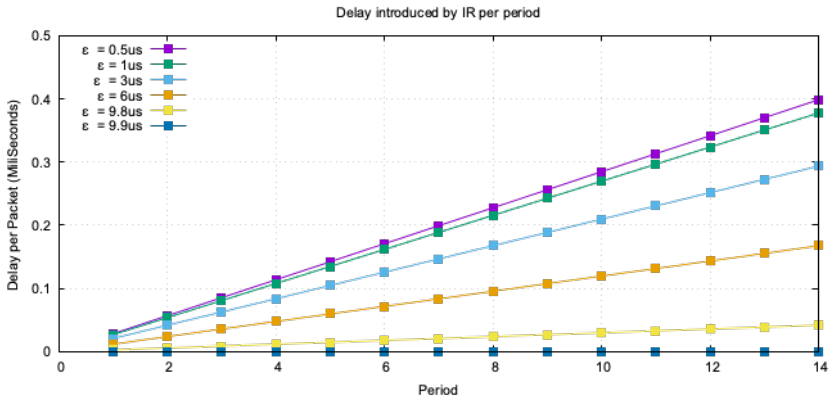


Figure 4.4: Delay of the first packet in period k for flow 1 with different ε

4.1.2 Different frequency offset values s_1

Different values for the frequency offset are simulated. When modifying the value for s_1, the interval, the period and the x_j values need also to be modified, as seen in figure 3.11.

When simulating for different values of s_1, the introduced delay for the first packet of the kth period of the first source just suffers an small deviation from the values obtained in the last section. For example, the introduced delay per period in the case of simulating s_1 with a value of 1.1, is $28.5\mu s$, in contrast for the case of $s_1 = 1.001$ is $28.47\mu s$. The results slightly vary and graphically is almost impossible to appreciate the difference from figure 4.1. In order to show more meaningful results, the divergence rate of the delay introduced by the ATS per second is calculated. Theoretically, the divergence is given by the following equation:

$$\text{div} = \frac{\text{Increased delay per period}}{\text{Period}} = \frac{3I\left(1-\frac{1}{s_1}\right)-3\varepsilon}{\frac{3I}{s_1}+3\varepsilon} \tag{4.1}$$

Figure 4.5 shows the results obtained in the simulations for different values of s_1 for source 1. The height of each step represents the divergence rate per second and the values are shown in table 4.3. As we can observe the divergence rate of the delay introduced by the ATS in each second for source 1, increases when s_1 increases, obtaining bigger divergence rates with bigger values of s_1.

	$s_1 = 1.001$ $\varepsilon = 0.5\mu s$	$s_1 = 1.05$ $\varepsilon = 0.5\mu s$	$s_1 = 1.1$ $\varepsilon = 0.5\mu s$	$s_1 = 1.2$ $\varepsilon = 0.5\mu s$	$s_1 = 1.2$ $\varepsilon = 5ns$
Divergence Rate (s)	0.000949901	0.047381546	0.094527363	0.18809901	0.19986001

Table 4.3: Divergence rate per second

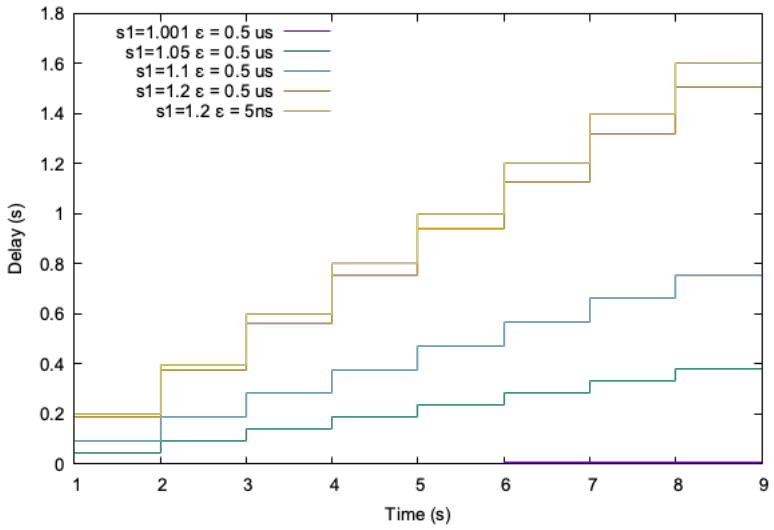


Figure 4.5: Divergence rate per second for flow 1 with different s_1 values

Notice that the divergence rate also depends on the ε value, which is the interval time between packets of different sources. The closer packets between sources are, the bigger the divergence rate is. This is due to the reduced time interval between packets, achieving its maximum rate when $\varepsilon \to 0$. Theoretically, the largest divergence rate for each s_1 value is, as shown in [20, Proposition 10]:

$$\begin{aligned} \lim_{\varepsilon \to 0} \text{div} &= \frac{nI\left(1-\frac{1}{s_1}\right)}{\frac{nI}{s_1}} \\ &= s_1 - 1 \end{aligned} \tag{4.2}$$

The simulation shows that the divergence rate when a value $s_1 = 1.2$ and an $\varepsilon = 5ns$ are considered, is bigger than the divergence rate for an $s_1 = 1.2$ and $\varepsilon = 5\mu s$. With an $\varepsilon = 5ns$ the divergence rate is equal to $div = 0.19986001$, which is almost equal to the theoretical value $div = s_1 - 1 = 0.2$.

The simulation clearly proves that the divergence rate of the delay through the ATS increases when s_1 is bigger and ε lower, having a maximum value when $\varepsilon \to 0$. The value for s_1 has been selected as $s_1 \leq \sqrt{\rho}$, being ρ the frequency offset between the clock of source 1, $\mathcal{H}_1$, and the clock of the ATS, $\mathcal{H}_{ATS}$. The worst-case delay per second in the network increases at most $\sqrt{\rho} - 1$ for each ρ value regardless the synchronization accuracy used for clock synchronization in the network.

4.2 Extension of the proof

One of the drawbacks of the scenario presented in the last section, is that it considers the FIFO system as ideal, this is, it has an infinite service curve. This assumption does not resemble to the case of a real network where we have different data rates at the output of the network elements. Therefore, the question that should be answered now is if this proof still holds if we consider different service curves in the FIFO system. The following scenario is proposed:

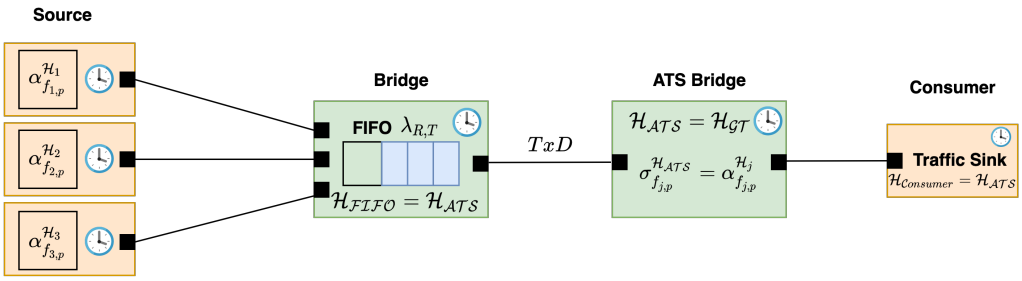


Figure 4.6: Scenario model with different service curves for the FIFO system

4.2.1 Mathematical reasoning

Appendix E gathers all the calculations performed for the extension proposal. In this section the main results are presented.

The results obtained in appendix E are divided in three cases, depending on the values of the transmission delay and always considering that the FIFO system is stable, as studied in the appendix. The values for the delay suffered through the IR by the first packet of the kth period of the first source are calculated. We assume that other packets exist in the network before $A^1_{j,1}$, thus, for $k = 1$ $D^1_{j,1} - A^1_{j,1} = 0$ is considered in the 3 studied cases.

Case 1

- Considering $TxD > \varepsilon$ and $TxD > I(1 - \frac{1}{s_1})$

$$\forall k \neq 1 \in \mathbb{N} \quad D^1_{1,k} - A^1_{1,k} \geq (n(k-1)-1)(I(1-1/s1)-\varepsilon) \tag{4.3}$$

Case 2

- Considering $TxD > \varepsilon$ and $TxD < I(1 - \frac{1}{s_1})$

$$\forall k \neq 1 \in \mathbb{N} \quad D^1_{1,k} - A^1_{1,k} \geq n(k-2)(I(1-1/s1)+\varepsilon) - TxD - (n-1)\varepsilon + nI(1-1/s_1) \tag{4.4}$$

Case 3

- Considering $TxD < \varepsilon$

$$\forall k \neq 1 \in \mathbb{N} \quad D^1_{j,k} - A^1_{j,k} \geq (k-1)n\left(I\left(1-\frac{1}{s_1}\right)-\varepsilon\right) \tag{4.5}$$

Mathematical results show that the delay is completely unbounded and diverges as k increases for all the cases. Furthermore, the delay is increased each period a value equal to $I(1-s_1)+\varepsilon$, which is the same as in the original proof. The obtained results prove that the instability is shown with any data rate if the stability condition of the FIFO system is met. The results obtained in the simulations are discussed in the following section.

4.2.2 Results of the simulation

4.2.2.1 Worst-case delay

Introducing a service curve in the FIFO system enables to study the worst-case delay of the network, which in the case of the scenario model proposed, is equal to the worst-case delay of the FIFO system. As explained in chapter 2, regulators are studied assuming the shaping for free property. Recall that the shaping property states that the wort-case delay of the network is never increased by the regulators. Therefore, the worst-case delay of the FIFO system should never be increased by the ATS. In this section, the theoretical worst-case delay is computed and by simulation the end-to-end delay is obtained, revealing that the shaping for free property does not hold in this scenario.

To calculate the worst-case delay of the network, we rely on figure 4.8. We know consider that each flow has two different and valid arrival curves, as seen in figure 4.7. On the one hand, the red line represents the arrival curve configured in the ATS scheduler (always taking into account the variation due to the header overhead). The arrival curve is characterized by a rate $r = L/I$ and a burst size $b = L$. On the other hand, the green line represents the long-term arrival curve for the flow, which is also a valid arrival curve. Each source outputs two packets every period τ, which leads to an arrival curve with a lower rate $r = 2L/\tau$ but a higher burst $b' = 2L - rI$. The burst b' is calculated considering that at time $t = I$, the number of packets output ($2L$) has to be equal to $2L = b' + Ir'$.

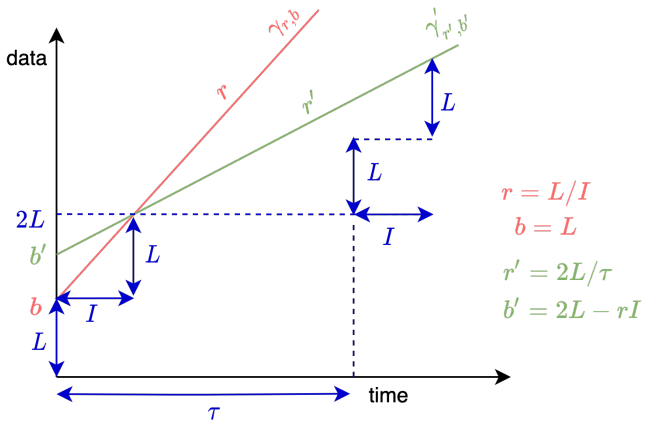


Figure 4.7: Arrival curves of the flow

As explained in [31, Section 1.2], when considering two arrival curves with parameters (r,b),

an arrival curve $\alpha(t)$ equal to the minimum of both of them can be obtained, which is

$$\alpha(t) = min(M + pt, b + rt) \tag{4.6}$$

where $M = L$ is understood as the packet size, $p = r$ as the peak rate, $b = b'$ as the burst size and $r = r'$ as the sustainable rate, as seen in figure 4.8.

Following the results obtained in [31, Proposition 1.4.1], the maximum delay for the flow is bounded by

$$D_{max} = \frac{M + \frac{b-M}{p-r}(p-R)^+}{R} + T \tag{4.7}$$

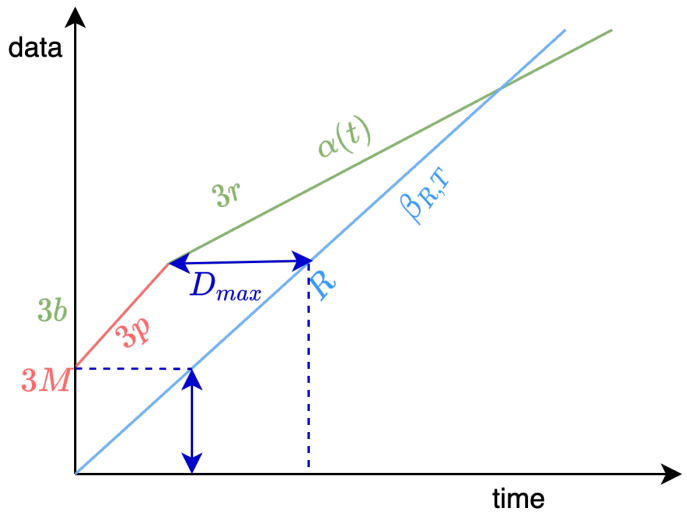


Figure 4.8: Computation of the delay bound

Consider the following simulation with the parameters shown in table 4.4 and $R = 437.5$ KBPS. In order to obtained the arrival curve of the three flows, the arrival curve of each flow is added, as seen in figure 4.8. Using equation 4.7 and $3M = 4548$ (bytes), $3b = 6061.119899$ (bytes), $3p = 454.8$ KBPS and $3r = 303.4880101$ KBPS, a worst-case delay equal to $D_{max} = 10.79085714ms$ is obtained. The theoretical worst-case delay is a conservative value, however is still an upper bound and no packet should violate that value.

The figure 4.9 shows the results obtained. The red line represents the theoretical worst-case delay for flow 1 and the blue dots represent the end-to-end delay that each packet of flow 1 suffers. This figure shows clearly that under these conditions, the shaping for free property does not hold and the worst-case delay is increased by the ATS.

Time model	**Source 1**	**Source 2**	**Source 3**
Δ (μs)	1	1	1
s_1	1.001	1.001	1.001
Interval I (ms)	10	10	10
ε (μs)	0.5	0.5	0.5
Period τ (ms)	29.97153	29.97153	29.97153
$x_j(ms)$	5	14.990001	24.98002
Traffic model	**Flow 1**	**Flow 2**	**Flow 3**
Peak Data Rate (KBPS)	147	147	147
Packet Size (bytes)	1470	1470	1470
Traffic type	UDP	UDP	UDP
ATS scheduler	**Flow 1**	**Flow 2**	**Flow 3**
Data Rate (KBPS)	151.6	151.6	151.6
Burst Size (bytes)	1516	1516	1516

Table 4.4: Simulation parameters

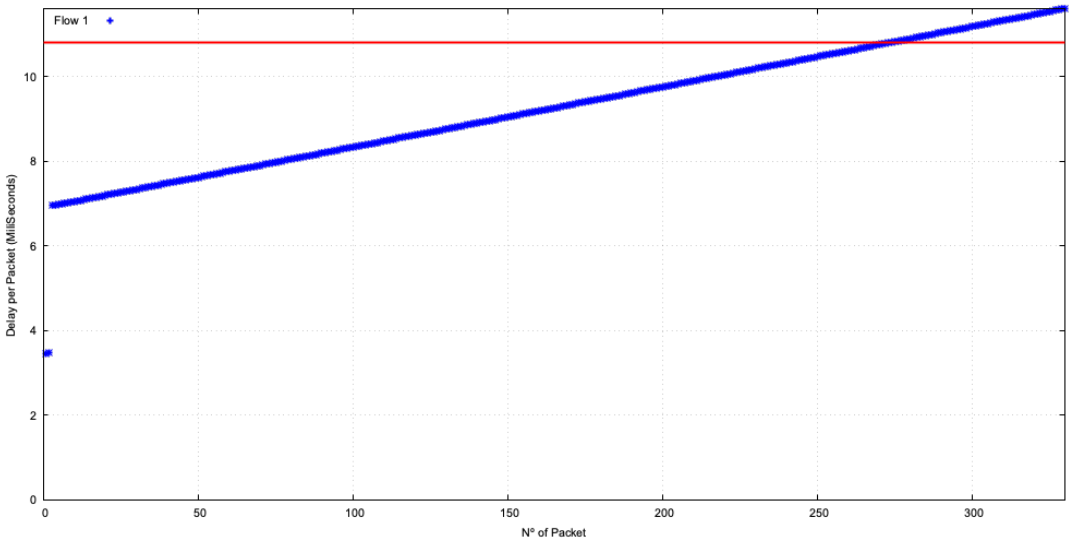


Figure 4.9: End-to-end delay and worst-case delay

4.2.2.2 Different data rates

The second simulation carried out considers a wide variety of different data rates. The values for the simulation are the same ones shown in table 4.1 and the results plotted in figure 4.10. The figure shows the introduced delay by the ATS for the first 5 periods considering different data rates of the FIFO system.

When 10 kbps are consider at the output rate of the FIFO system, the FIFO is completely

unstable. Because of that, the introduced delay of the ATS is 0. However, the end-to-end delay is completely unbounded. For rates between 100 kbps and 100 Mbps we are in the cases 1 and 2 of the mathematical proof. The introduced delay for the first period is a bit lower than the case considering an infinite service curve. Nevertheless, the introduced delay per period is the same and diverges at the same rate. Finally, for rates between 1 Gbps and 100 Gbps, the service curve behaves as if it were an infinite service curve, obtaining the same results as with the original proof. As we can see, the introduced delay per period is equal in all the cases, which means that the divergence rate per second is equal to the case studied in the last section. Simulation shows that if the FIFO system is stable, no matter which data rate is selected, the introduced delay by the ATS is unbounded, which proves the instability.

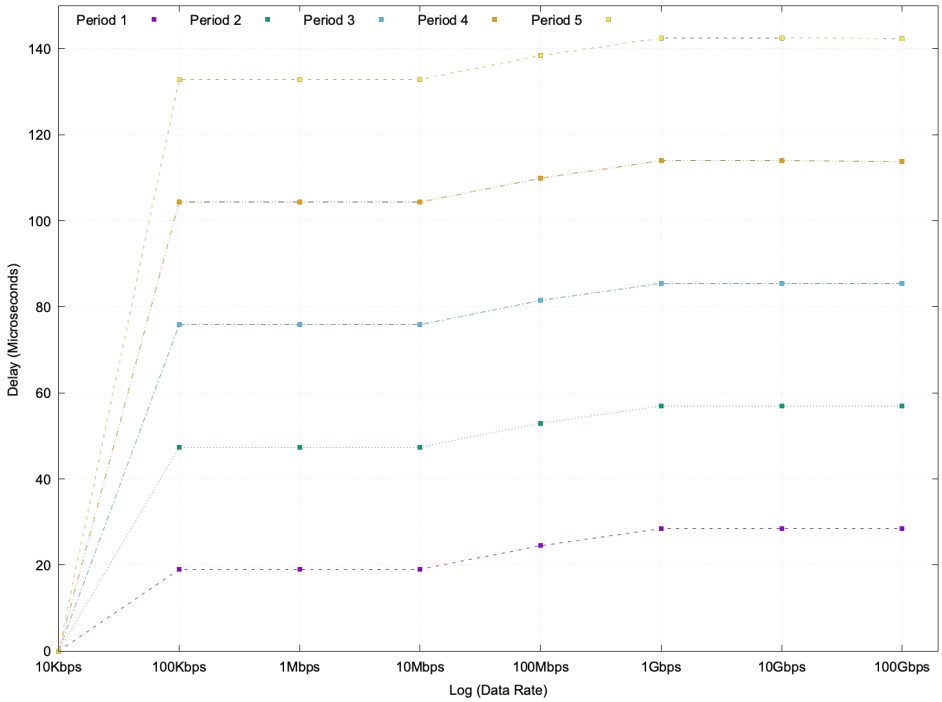


Figure 4.10: Delay per period introduced by the ATS for flow 1 with different data rates

The simulation results corroborate with a theoretical proof proposed in [20] for ATS within synchronized networks. Furthermore, the results extend the ones of the original proof. Lastly, it allows to show that the different modules developed within the project can be assembled together, performing a new wide variety of simulations that extend the capabilities of ns-3.

Chapter 5

Conclusion

The final simulations carried out along this thesis prove that ATS or IR can lead to system instability when considering specific traffic and clock models. Time-Sensitive Networks aim to provide bounded delays throughout the network. However, the existence of the adversarial proof shows that the clock deviations between network elements can lead to an unbounded delay. Furthermore, the extension of proof demonstrates that an arbitrary data rate can be selected at the output of the FIFO system so that the instability is maintained in any case. This result extends the original proof, making the results more meaningful. Due to the safety-critical communications, strong certification levels are applied in time-sensitive networks. Therefore, the existence of this adversarial proof highlights the need of finding solutions to this problem.

Aside from the final simulation, different modules have been developed within this thesis. The local clock module developed in section 3.1 allows to introduce the concept of local times in ns-3. It provides an open interface where different clock models can be attached. This new feature extends the capabilities of ns-3, enabling a new wide variety of simulations, not only suitable for Time-Sensitive Networks. The independent designed module co-exists with the already existing modules of ns-3 and requires no change of the existing code of the simulator, making it suitable for any user.

Furthermore, the ATS module proposed in section 3.2 introduces the concept of shaping in ns-3, as proposed by the IEEE Time-Sensitive Networks 802.1 working group. The module built on the top of the traffic control layer of ns-3 provides the flexibility and modularity needed in each simulation. It can be attached to the output port of a bridge and can be configured with the desired parameters. Also, it can be easily extended to implement more complex structures with different kinds of schedulers.

Finally, many tests and examples have been developed in order to validate the modules, as explained in appendix A and D. As well as the final simulation itself, which confirms the well designed and correct iterations with other ns-3 modules and the achievement of the different goals proposed for the thesis.

5.1 Future work

The master thesis had the objective of simulating the proof proposed in chapter 2. This being said, there is still a lot of margin to improve the simulation. Some fields of further study could be:

- Simulate in ns-3 a real synchronized network. In this project each clock model is bounded by the synchronization precision parameters. However, it does not exist a synchronization protocol running between nodes. The clock module takes into account this possibility, providing an interface for clock updates when synchronization messages arrive to the nodes. A future implementation could develop a module including the IEEE 1588 synchronization protocol, as proposed by the IEEE TSN working group.

- Implement more realistic clock models. The clock module opens the possibility to attach clock models to each node, nevertheless, the clock models need to be designed. This is not an easy task due to the difficulty of modelling clocks. However, in the present project, an interface to attach a clock model without the need of rewriting the exiting ns-3 code is proposed, facilitating future implementations of clock models in ns-3.

- As mentioned before, due to the requirements of Time-Sensitive Networks, the instability problem needs to be faced. In the present project, no solutions are proposed. However, some solutions have emerged to cope with the unbounded delay problems in ATS. One of the solutions proposes to manage the constrains of the ATS, adapting the committed data rate of each shaper to the clock deviation. A natural step of the present project would be to simulate the solution in the present scenario and evaluate the performance. Moreover, an extension of the scenario proposed in this thesis could be simulated, having more network elements within the network and achieving more meaningful results.

- Extend the capabilities of ns-3 in terms of Time-Sensitive Networks. TSN are gaining a lot of importance in the field of critical communications, thus implementing and extending the capabilities of the simulator in this field would help future researchers.

Chapter 6

Budget

The total cost of the thesis accounts for the hardware cost and human resources cost. The human resources cost has been established as the cost of a trainee engineer: 12€/h

Tool	**Cost €**
Computer	1000
TOTAL	1000

Human resources	**Hours**	**Cost €**
Research	150	1800
Software development	400	4800
Test and analysis	250	3000
TOTAL	800	9600

Total cost for the thesis: 10600€.

Bibliography

[1] *802.1AS-2011 - IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*. URL: https://ieeexplore.ieee.org/document/5741898.

[2] *802.1Qbv - Enhancements for Scheduled Traffic i.e Time-Aware Shaper*. URL: http://www.ieee802.org/1/pages/802.1bv.html.

[3] Matthew Andrews. "Instability of FIFO in the permanent sessions model at arbitrarily small network loads". In: *ACM Transactions on Algorithms* (Jul. 2009). URL: https://dl.acm.org/doi/pdf/10.1145/1541885.1541894.

[4] *Asynchronous Traffic Shaping drafts*. URL: http://grouper.ieee.org/groups/802/1/pages/802.1cr.html.

[5] Ehsan Mohammadpour Eleni Stai Maaz Mohiuddin Jean-Yves Le Boudec. "Latency and Backlog Bounds in Time-Sensitive Networking with Credit Based Shapers and Asynchronous Traffic Shaping". In: *2018 30th International Teletraffic Congress (ITC 30)* (Sept. 2018). URL: https://ieeexplore.ieee.org/document/8493026.

[6] Jean-Yves Le Boudec. "A Theory of Traffic Regulators for Deterministic Networks With Application to Interleaved Regulators". In: *IEEE/ACM Transactions on Networking (Volume: 26 , Issue: 6 , Dec. 2018)* (Nov. 2018). URL: https://ieeexplore.ieee.org/document/8519761.

[7] *Clock and ATS modules*. URL: https://github.com/guikoala/ns3-cpn.

[8] Anne Bouillard Marc Boyer Euriell Le Corronc. "Deterministic Network Calculus: From Theory to Practical Implementation". In: (Oct. 2018). URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9781119440284.

[9] G.810. *ITU : Definitions and terminology for synchronization networks*. 2004. URL: https://www.itu.int/rec/T-REC-G.810-199608-I/en.

[10] IEEE. "IEEE Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology—Random Instabilities". In: *IEEE Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology—Random Instabilities* (Feb. 2009).

[11] IEEE. "IIEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems". In: *IEEE Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology—Random Instabilities* (Jul. 2008).

[12] Arne Neumann Lukasz Wisniewski Rakash SivaSiva Ganesan Peter Rost Jürgen Jasperneite. "Towards integration of Industrial Ethernet with 5G mobile networks". In:

2018 14th IEEE International Workshop on Factory Communication Systems (WFCS) (Jun. 2018). URL: https://ieeexplore.ieee.org/abstract/document/8402373.

[13] Jinoo Joung. "Regulating Scheduler (RSC): A Novel Solution for IEEE 802.1 Time Sensitive Network (TSN)". In: *Electronics 2019* (Feb. 2019). URL: https://doi.org/10.3390/electronics8020189.

[14] Dong-Won Park Swaminatha Nataraja Arkady Kanevsky. "Fixed-priority scheduling of real-time systems using utilization bounds". In: *Journal of Systems and Software Volume 33, Issue 1, April 1996, Pages 57-63* (Feb. 1999). URL: https://doi.org/10.1016/0164-1212(95)00105-0.

[15] Tatsuya Maruyama Tsutomu Yamada Shouji Yoshida Mitsuyasu Kido Chikashi Komatsu. "NS-3 based IEEE 1588 synchronization simulator for multi-hop network". In: *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)* (Oct. 2015).

[16] Jin Seek ChoiBum Sik BaeHyeong Ho LeeHyeung Sub Lee. "Round-Robin Scheduling Algorithm with Multiple Distributed Windows". In: *International Conference on Information Networking ICOIN 2002: Information Networking: Wireless Communications Technologies and Network Applications* (Sep. 2002). URL: https://link.springer.com/chapter/10.1007/3-540-45801-8_76.

[17] *Linux Token Bucket Filter*. URL: https://linux.die.net/man/8/tc-tbf.

[18] *Linux Traffic Control*. URL: http://tldp.org/en/Traffic-Control-HOWTO/ar01s04.html.

[19] *LocalClock module PDF*. URL: https://github.com/guikoala/Clock-per-node-documentation.

[20] Jean-Yves Le Boudec Ludovic Thomas. "On Time Synchronization Issues in Time-Sensitive Networks with Regulators and Nonideal Clocks". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* (Jun. 2020). URL: https://dl.acm.org/doi/10.1145/3392145.

[21] *Massif visualizer tool*. URL: https://github.com/KDE/massif-visualizer.

[22] *Network Time Protocol Version 4: Protocol and Algorithms Specification*. URL: https://www.rfc-editor.org/info/rfc5905.

[23] *NS-3 a discrete-event network simulator for internet systems*. URL: https://www.nsnam.org.

[24] *NS-3 Real Time Simulator Manual*. URL: https://www.nsnam.org/docs/manual/html/realtime.html.

[25] *P802.1DG – TSN Profile for Automotive In-Vehicle Ethernet Communications*. URL: https://1.ieee802.org/tsn/802-1dg/.

[26] *Posix Linux clocks*. URL: https://linux.die.net/man/3/clock.

[27] Zhou Zifan Yan Ying Berger Michael Stübert Ruepp Sarah Renée. "Analysis and Modeling of Asynchronous TrafficShaping in Time Sensitive Networks". In: *Proceedings of 2018 14th IEEE International Workshop on Factory Communication Systems* (Jun. 2018). URL: https://doi.org/10.1109/WFCS.2018.8402376.

[28] *Repository with the code for the merge request*. URL: https://gitlab.com/nsnam/ns-3-dev/-/merge_requests/332.

[29] Matthieu Coudron Stefano Secci. *Per node clocks to simulate time desynchronization in networks*. URL: https://www.nsnam.org/workshops/wns3-2016/posters/per-node-clock-abstract.pdf.

[30] Mitsuyasu Kido Chikashi Komatsu Tatsuya Maruyama Tsutomu Yamada Shouji Yoshida. "NS-3 based IEEE 1588 synchronization simulator for multi-hop network". In: *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)* (2015). URL: https://ieeexplore.ieee.org/document/7324691.

[31] Jean-Yves Le Boudec Patric Thiran. "A Theory of Deterministic Queuing Systems for the Internet". In: (Version December 13, 2019). URL: https://ica1www.epfl.ch/PS_files/netCalBookv4.pdf.

[32] "Time-Sensitive Networking Profile for Industrial Automation". In: *IEC/IEEE 60802* (2019). URL: https://1.ieee802.org/tsn/iec-ieee-60802/.

[33] *TSN overview*. URL: http://grouper.ieee.org/groups/802/1/files/public/docs2017/tsn-farkas-intro-0517-v01.pdf.

[34] *Valdgrind tool for memory management and profiling*. URL: https://valgrind.org.

Appendix A

Local time simulator implementation example

The *Clock* model validation has been done using both, examples and test. Unitary tests have been written to verify the internals of *LocalTimeSimulatorImpl* and *LocalClock* classes using *PerfectClockModelImpl*. They can be found in src/test/clock-test.cc. Here we present the main characteristics of the two-clocks-simple.cc available in src/clock/example.

The scenario model is shown in figure A.1 and the simulation parameters in table A.1.

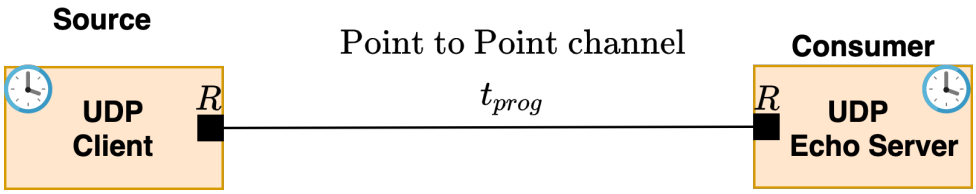


Figure A.1: Simulation Scenario

Scenario Settings	**Node 1**	**Node 2**
Application	UDPClient	UDPEchoServer
Interval Time (s)	3	
Net Device	Point to Point	Point to Point
Data Rate	5mbps	5mbps
Packet Size (bytes)	1024	1024
Propagation Delay	2ms	2ms
Simulation (s) Time	100	100
Clock	Non-ideal	Ideal

Table A.1: Validation scenario parameters

The scenario model uses two nodes connected by point to point Net Devices. *UdpClient* and *UdpEchoServer* applications have been installed in the nodes to generate the traffic. The client runs a non-ideal clock using the *PerfectClockModelImpl* class. This class allows to define the time relative function between the global time and the local time. The time relative function, shown in figure A.2, is defined by a set of affine functions with a frequency offset (the slope of the function), as well as an initial offset (initial value when global time equals 0). When the frequency offset is bigger than one, the local clock runs faster. On the other hand, when the frequency offset is below one, the local clock runs slower. Likewise, when it is set to one, the clock runs at the same speed as the global clock and it is considered to be a perfect clock.

In order to build the shape of the function, several clock updates are triggered. Each 20 seconds, the clock update changes the frequency offset between clocks. In order to provide continuity in each updated point, the initial offset of the function has to be re-calculated. The time relative function could have been done using a single clock model, without the need of updating the clock each time. As an example, we present the *PeriodicClockModel*, which can be found in "src/model". This model, represents a periodic clock, with different frequency offsets changing every predefined interval of time. Conversely, the server node runs a perfect clock.

In order to make the *clock* model run in the example, few changes need to be done in the main file. Remark that the *clock* module can run with the preexisting code of ns-3. Few changes need to be done while coding the scenario model in the main file. The changes are as follows:

• First, the global variable *SimulatoImplementationType* needs to be changed, pointing to the *LocalSimualtorImpl*.

```
    GlobalValue::Bind ("SimulatorImplementationType", StringValue ("ns3::LocalTimeSimulatorI
```

• Second, The *ClockModel* implementation has to be created. In the case of the example, we use the *PerfectClockModelImpl*.

```
Ptr<PerfectClockModelImpl> clockImpl = CreateObject <PerfectClockModelImpl> ();

clockImpl -> SetAttribute ("Frequency", DoubleValue (freq));

clockImpl -> SetAttribute ("Offset", TimeValue (init_offset));
```

• Third, create the *LocalClock* object, set the attribute *m_clock* and aggregate it to the node.

```
Ptr<LocalClock> clock = CreateObject<LocalClock> ();

clock -> SetAttribute ("ClockModelImpl", PointerValue (clockImpl));

node -> AggregateObject (clock);
```

Figure A.3 shows the result obtained in the simulation. We represent the time at which the client node sends packets. As it can be appreciated, the time at which packets are sent variate slightly depending on the frequency offset between clocks. To compare the performance, we plot the same simulation, but without using the *clock* module. We can see how the clock on the client

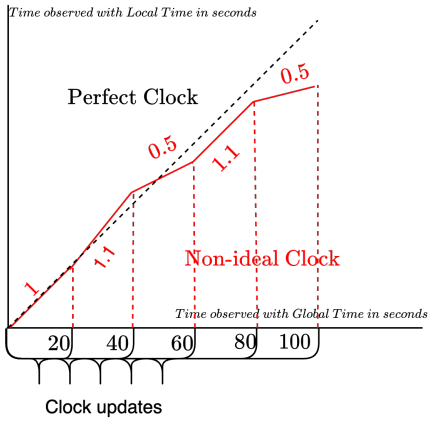


Figure A.2: Ideal and Non-Ideal clock model

tends to go slower even though there are some intervals of times where it goes faster.

Between seconds 0 and 20, both clocks have the same frequency offset. However, after second 20, a clock update is triggered. This clock update modifies immediately the packets that are scheduled. As it can be seen, straight after second 20, the slope starts moving downwards, having the client clock running faster. The interval between packets changes from 3 to 2.7 seconds. Immediately after second 40 the slope moves upwards moving the interval from 2.7 to 6 seconds. As we can see, the frequency difference directly affects at which time nodes send packets.

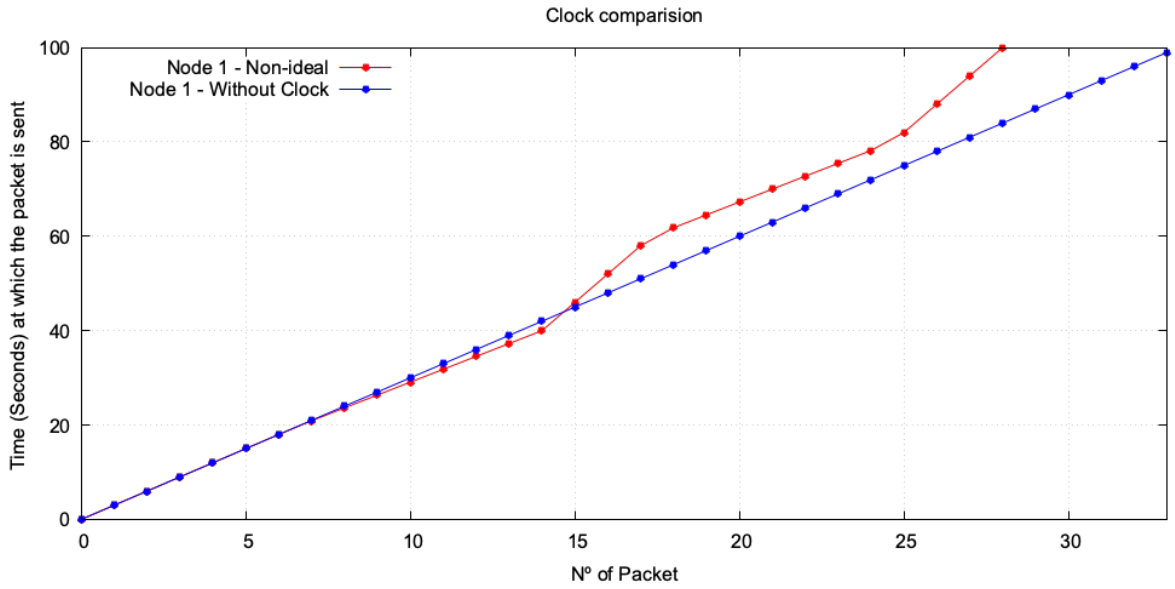


Figure A.3: Results obtained for Ideal and Non-Ideal clocks

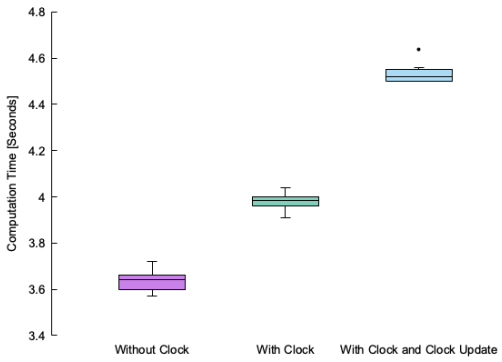
Appendix B

Performance evaluation and limitations of clock module

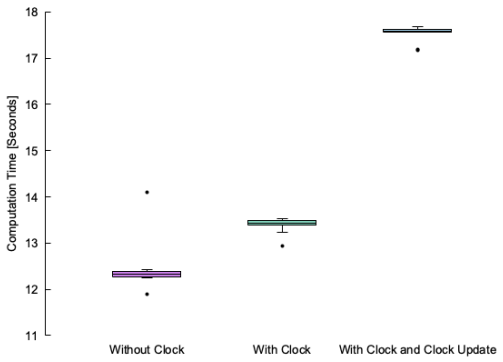
For the purpose of evaluating the overhead added by the module, the computation time has been taken into account. A flexible scenario has been considered, where the parameters, such as the number of nodes and the clock models, can be tuned. Half of the nodes in the network communicate with the other half through a switch. The application generates random data flows within some intervals of time. Nodes can be tuned without clocks, with clocks and with clock updates, in such a way that the three possible scenarios are tested. Mention that clock updates affect all the nodes of the network. Simulations have been performed with 50 nodes B.1a, 100 nodes B.1b, 200 nodes B.1c and 500 nodes B.1d.

The PC used for this testing uses an Intel Core i5 6200U Processor with 8GB RAM and Ubuntu 20.04. Ten simulations for each scenario have been carried out in order to have an average value of the computation time. In order to compute the values, *Time* linux command has been used.

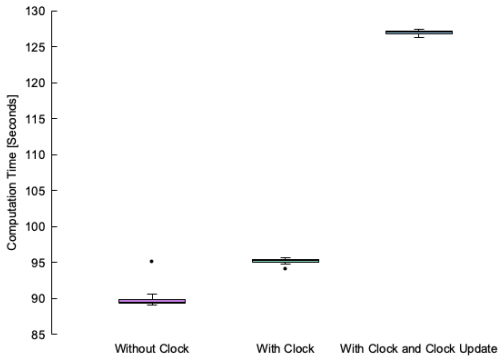
As we could expect, the computation time increases with the number of nodes, as well as with the different configurations of the clocks. Simulations show that in the case of using clocks, the increment in time is between 3% and 9%, with respect to the simulation without clocks. However, when using frequent clock updates, the simulation time skyrockets from 3% to 40% for simulations with bigger number of nodes. As we can see, this increase in time can be a drawback with big and complex scenarios that require constant clock updating and more computing resources.



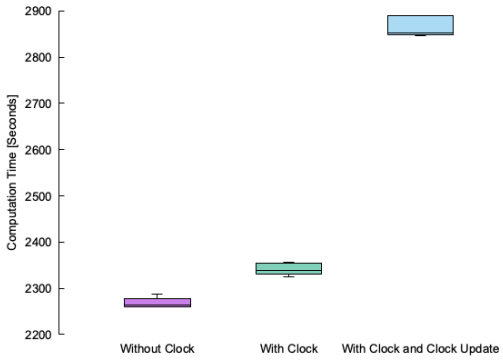
(a) Simulation with 50 nodes



(b) Simulation with 100 nodes



(c) Simulation with 200 nodes



(d) Simulation with 500 nodes

Figure B.1: Results on computation time

Appendix C

ATS UML design

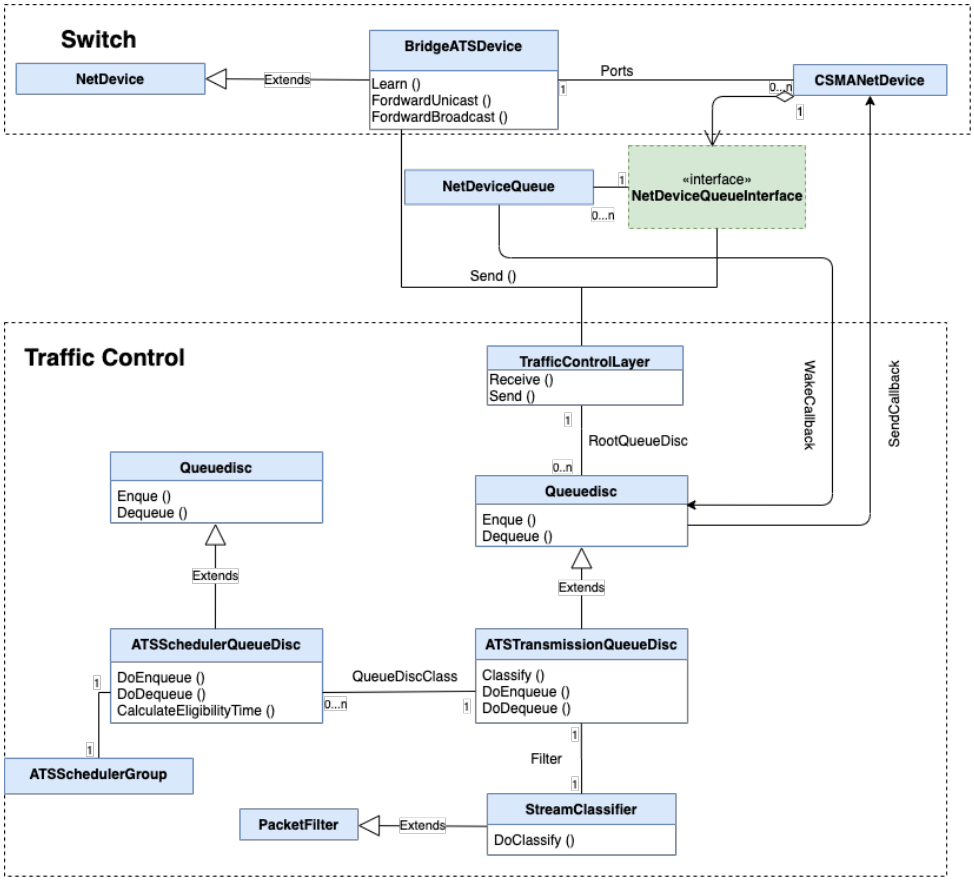


Figure C.1: ATS design UML

Appendix D

ATS validation

The ATS module validation has been done using both examples and performing low level tests. In this appendix, we present the example carried out and the main results obtained. The following scenario model is presented:

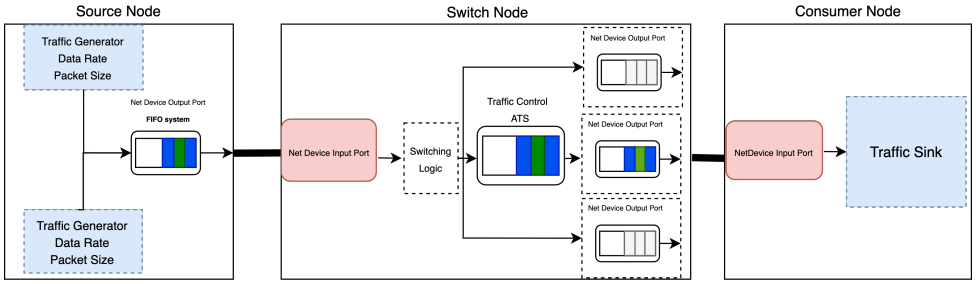


Figure D.1: ATS validation scenario

The source node contains two applications that generate the traffic of the source. The application used is the *OnnOffApplication* that ns-3 provides. This application generates constant packets in every *on* period and stops sending packets in the *off* periods. In the case of the simulation explained in this appendix, the *off* periods are tuned to zero, such that the application generates all the time packets at a constant rate. In order to validate the ATS model, first we need to create a bursty traffic, in such a way that the ATS shaping mechanism reshapes the traffic into the original burst size. In order to create the bursty traffic we need to place an aggregate scheduler shared by the different flows. The most common aggregate scheduler is the FIFO system. By default, the *NetDevice* object of ns-3 provides a *Drop-tail* queue in the output port, which is a FIFO that drops frames when the queue starts building-up. Each time the frame is transmitted to the channel, the frame is enqueued in the FIFO system. Moreover, we use *CSMA* devices. These devices enter in random back-offs each time the channel is busy, increasing the burtiness at the output port.

The switch node contains the switching logic, as well as the traffic control system. The traffic

control system will implement the ATS functionalities, allowing to reshape the traffic before sending it.

The consumer node has the functionally of a traffic sink, which allows to calculate end to end delays in the scenario.

We propose three different variants of this scenario in order to validate the complete functioning of ATS.

D.1 Case A

In case A, we propose to generate two different flows in order to create contention in the output port of source node. Only flow 1 is sent to the consumer crossing the ATS, as shown in D.2. The flow settings are shown in table D.1. The differences between the flow settings in the application and in the ATS are due to the need to account for the headers inserted when processing the frames at layer 2. Thus, in order to achieve the same burst size than the sources, the parameters in the ATS must be modified.

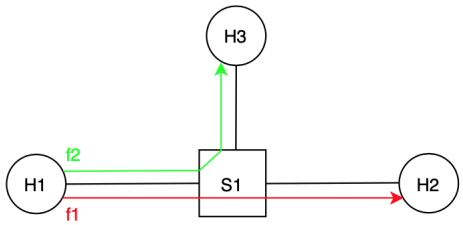


Figure D.2: Case A scenario

Flow Settings	**Flow 1**	**Flow 2**
Data Rate (KBPS) Packet Size (Bytes)	10 512	10 512
Flow Settings in ATS	**Flow 1**	**Flow 2**
Data Rate (KBPS) Burst Size (Bytes)	10.8984375 558	

Table D.1: Flows settings Case A

The obtained results are shown in figure D.3. The green line represents the output at the source for flow 1. The traffic pattern is perfectly defined, the interval time between frames is maintained constant throughout the time. The orange line represents the arrival time of flow 1 to the ATS.

We can appreciate how the burstiness of the flow has increased, the interval time has changed due to the FIFO system placed at the output of the source. The blue line is the eligibility time of the frames, representing the time at which the frames are sent to the output port of the switch. We suppose that it does not exist any other traffic that arrives to the ATS before the first frame of flow 1, thus, the first frame is directly sent to the channel. As we can see, the ATS has shaped perfectly flow 1, recovering the burst size of the original flow. Therefore, we can confirm that the ATS is reshaping traffic depending of the data rate and burst size configured.

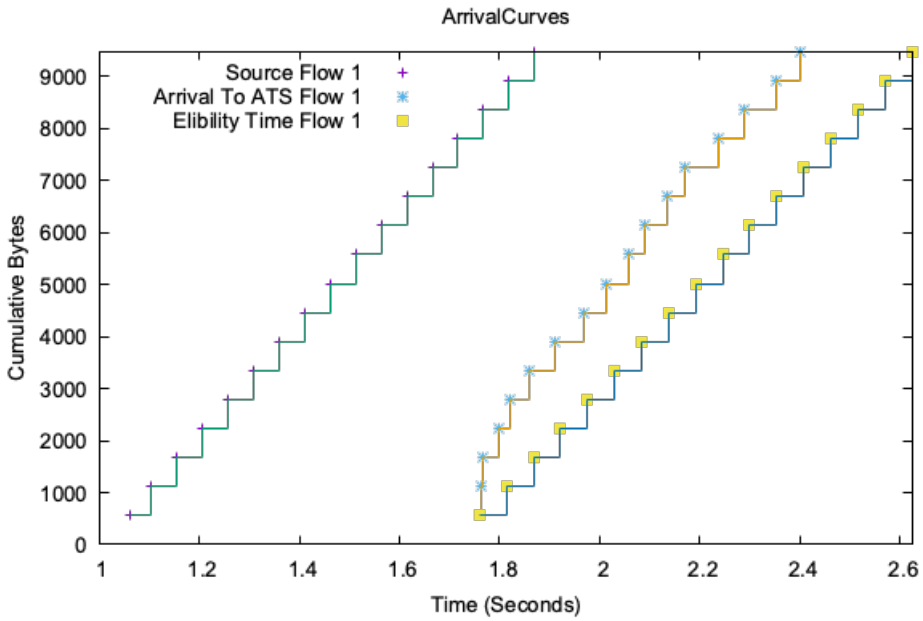


Figure D.3: Case A results

D.2 Case B

In this second case, we study the behaviour of the group eligibility time. This is a key property that allows to implement the behavioural model of an ATS or interleaved regulator. For this second case, we consider the scenario of figure D.4 and the parameters shown in table D.2.

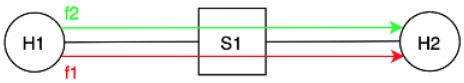


Figure D.4: Case B scenario

Flow Settings	Flow 1	Flow 2
Data Rate (KBPS)	50	40
Packet Size (Bytes)	512	512
Flow Settings in ATS	**Flow 1**	**Flow 2**
Data Rate (KBPS)	50	40
Burst Size (Bytes)	558	558

Table D.2: Flows settings Case B

During the setup of the parameters in the ATS, an error was made. The data rate is equal to 50KB and it's lower than the value that it's supposed to be. Therefore, the frames are delayed more in the ATS. However, in this case we are not focused on that, what we want to see is if the group eligibility time has been taken into account. For this purpose, we create two flows, both belonging to the same traffic class. Flow 1 is faster and is initialized before flow 2. The difference of data rates will allow flow 1 to insert more frames than flow 2, thus, flow 2 will have to be aware of the group eligibility time constantly updated by flow 1. Result obtained are shown in figure D.5.

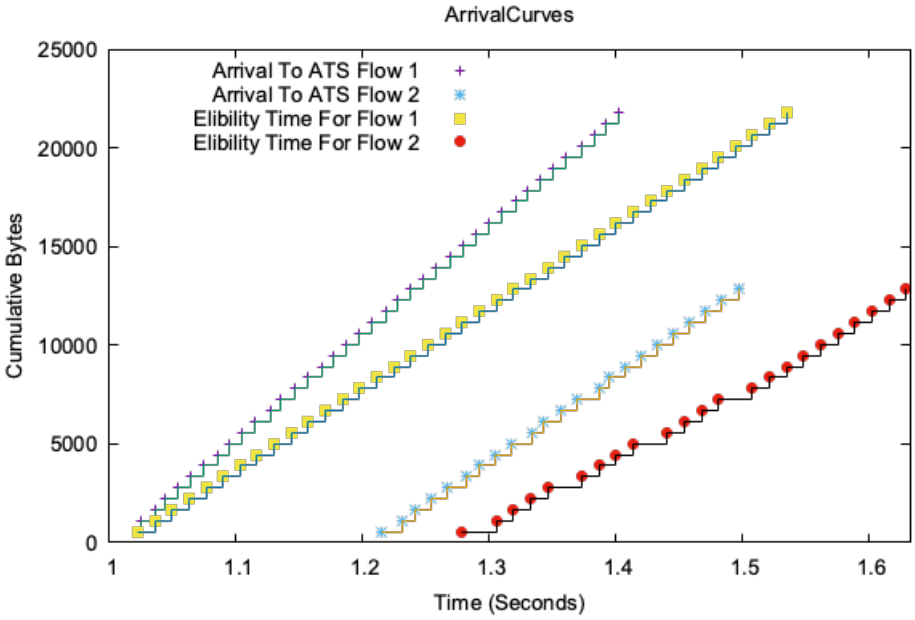


Figure D.5: Case B results

The arrival times to the ATS and the eligibility time for both flows are represented. The graphs show that flow 1 starts before flow 2. Because of the differences in the data rates, the arrival time and the eligibility time for flow 1 diverges. However, the green line and the blue line should be overlapped until the second 1.2, where second flow starts. At that time, the first frame of the

second flow arrives to the ATS. This frame should be directly sent to the channel because there are no other frames belonging to flow 2 before it, and the eligibility time should be equal to the arrival time of the frame. Nevertheless, the frame gets delayed because it's taking into account the group eligibility time set up by the flow 1. This pattern is repeated along the black line, where we can see that the interval time between red dots is bigger than the delay it's supposed to be and this is a direct consequence of the group eligibility time.

D.3 Case C

In case C, we study the delay introduced by the ATS in the scenario D.4, with the only purpose of verifying the shaping for free property. The ATS should never increase the worst-case delay of the network, i.e. when a frame gets excessively delayed by the network (worst-case delay) the ATS should never increase such delay, sending the frame directly to the output port. The parameters used for the simulation are shown in table D.3.

Flow Settings	**Flow 1**	**Flow 2**
Data Rate (KBPS)	80	80
Packet Size (Bytes)	512	512
Flow Settings in ATS	**Flow 1**	**Flow 2**
Data Rate (KBPS)	84.375	84.375
Burst Size (Bytes)	558	558

Table D.3: Flows settings Case C

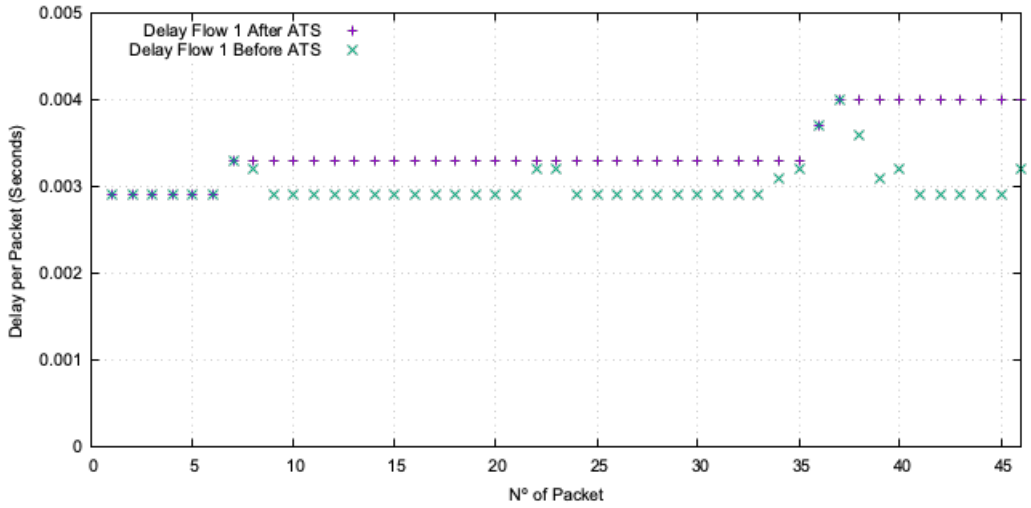


Figure D.6: Case C results

Figure D.6 shows the delay introduced by the ATS for each frame of the flow 1. This figure gives an intuitive vision of how the ATS performs. Green crosses represent the delay that the

frame has suffered before arriving to the ATS. In contrast, the purple crosses represent the delay that the frame suffers after the ATS. Likewise, the difference between the purple cross and the green cross is the delay introduced by the ATS. The first six frames arrive to the ATS without contradicting the parameters previously set up for flow 1. Because they do not contradict the constrains of the ATS, the frames are immediately released. However, the seventh frame suffers a bigger delay due to the containment in the previous FIFO queue. Because the interval time between the sixth frame and the seventh frame is even bigger than the regulation forced by the ATS, the frame is immediately released and there is not an extra delay introduced. For the case of the eighth frame, the delay before the ATS is reduced in comparison with frame number seven. Therefore, if the ATS releases immediately the frame, the interval time between packets would be reduced contradicting the constrains for the flow. To avoid this problem, the ATS introduces a small delay, forcing the interval time between frames to conform with the ATS parameters. The same happens for the next frames, where the ATS is forced to introduce some delay. As we can see, the worst-case delay is never increased. However, a penalty in terms of delay is paid.

Appendix E

Mathematical calculus for the proof extension

In this appendix, the mathematical proof for the extension proposed is explained.

For this analysis, we need to take into account the transmission delay due to the different service curves of the bridge with the FIFO system.

First, we impose the constraint that the FIFO system has to be stable. This assumption is a normal consideration when modeling the network, we select the output rate of the system such that the delay in the FIFO queue is not unbounded. To obtain this first condition, we apply some very basic concepts of network calculus. Each source is characterized by an arrival curve $\alpha(t)$ that can be shaped by a rate r and a burst size b. If we consider a conservative approach, each source outputs 2 packets every period. Therefore, the average rate of each source is equal to $r_{f_j} = 2L/\tau$, being L the packet size. The arrival curves of each source can be added obtaining a general upper-bound on the rate, $r = 2nL/\tau$. Using the definition of $\tau = nI/s1 + n\varepsilon$, we obtain an $r = \frac{2L}{I/s1+\varepsilon}$. To fulfil the FIFO stability condition, the output rate of the FIFO needs to be $R > \frac{2L}{I/s1+\varepsilon}$. The transmission delay (TxD) is defined as $TxD = L/R$, which imposes the condition $2TxD < I/s1 + \varepsilon$.

Three different cases are proposed depending on the value of the transmission delay.

Case 1

• $TxD > \varepsilon$, $2TxD < \frac{I}{s1} + \varepsilon$ and $TxD > I(1 - \frac{1}{s1})$

The traffic pattern is described relying in figure E.1. The time interval between $A^1_{j,1}$ and $A^2_{j,1}$ remains with a value equal to $\frac{I}{s1}$. Because $TxD < \frac{I}{s1}$, as soon as $A^2_{j,1}$ arrives to the FIFO queue, it is sent. However, packet $A^1_{j+1,1}$ suffers a queuing delay because $\varepsilon < TxD$. In such a manner that when $A^1_{j+1,1}$ arrives to the ATS the time between packets is not anymore ε, it is TxD. This will modify also the interval time between packets of the same flow $j+1$, which will change

from $\frac{I}{s1}$ to $\frac{I}{s1} - (TxD - \varepsilon)$, maintaining this pattern throughout the next periods.

The arrival time of $A^1_{j,k} \forall k \neq 1 \in \mathbb{N}$ to the ATS will be:

$$A^1_{1,k} \geq A^1_{1,0} + I/s1 + (n-1)(I/s1 - (TxD - \varepsilon)) + n(k-1)TxD + n(k-2)(I/s1 - (TxD - \varepsilon)) =$$
$$= A^1_{1,1} + nI/s1 + TxD(-(n-1) + n(k-1) - n(k-2)) + (n-1)\varepsilon + n(k-1)(I/s1 + \varepsilon) =$$
$$= A^1_{1,1} + nI/s1 + TxD + (n-1)\varepsilon + n(k-2)(I/s1 + \varepsilon)$$

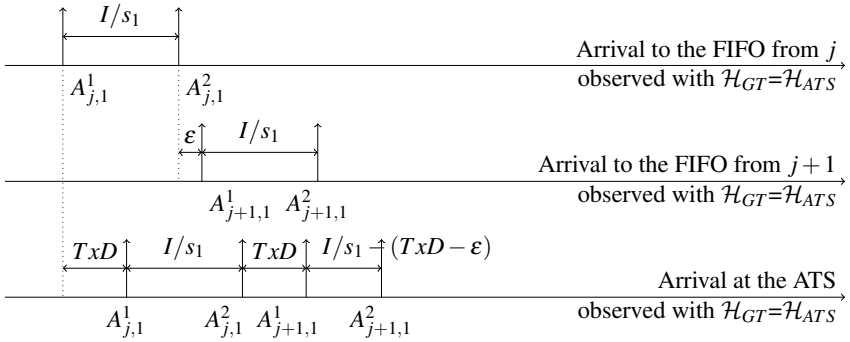


Figure E.1: Arrival time to the ATS when measured with $\mathcal{H}_{GT} = \mathcal{H}_{ATS}$

We compute now the time at which packets are released from the ATS, as shown in figure E.2. Packet $D^2_{j,1}$ is not blocked by the head of the line packet, because the TxD is bigger than the delay inserted by the ATS for packet $A^2_{j,1}$. For the remaining cases we observe that packets get blocked.

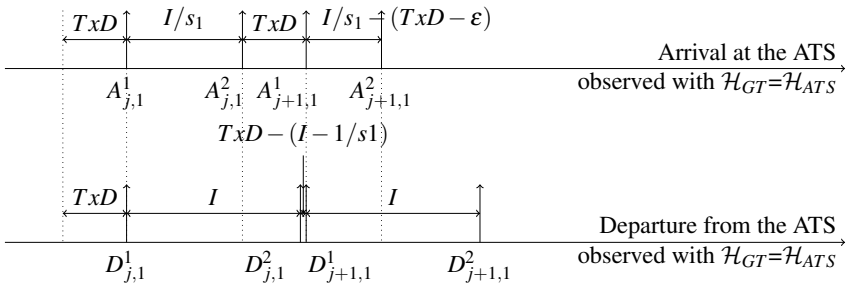


Figure E.2: Release time from the ATS when measured with $\mathcal{H}_{GT} = \mathcal{H}_{ATS}$

The departure time of $D^1_{j,k} \forall k \neq 1 \in \mathbb{N}$ from the ATS will be:

$$D^1_{1,k} \geq D^1_{1,1} + nI + TxD - I(1 - 1/s1) + nI(k-2)$$

The delay suffered through the IR by the first packet of the kth period, $\forall k \neq 1 \in \mathbb{N}$, of the first source is, when measured with $\mathcal{H}_{ATS} = \mathcal{H}_{GT}$:

$$
\begin{gathered}
D^1_{1,k} - A^1_{1,k} \geq D^1_{1,1} + nI + TxD - I(1-1/s1) + nI(k-2) - \\
-A^1_{1,1} - nI/s1 - TxD - (n-1)\varepsilon - n(k-2)(I/s1+\varepsilon) = \\
= (n-1)I(1-1/s1) - (n-1)\varepsilon + n(k-2)(I(1-1/s1)-\varepsilon) = \\
= (n-1)(I(1-1/s1)-\varepsilon) + n(k-2)(I(1-1/s1)-\varepsilon) = \\
= (n(k-1)-1)(I(1-1/s1)-\varepsilon)
\end{gathered}
$$

We consider that no other packets exist in the network before $A^1_{j,1}$, so $D^2_{j,1} - A^1_{j,1} = 0$.

Case 2

• $TxD > \varepsilon$, $2TxD < \frac{I}{s1} + \varepsilon$ and $TxD < I(1 - \frac{1}{s1})$

In this case, the arrival time of packets is the same as the studied in the case before. The only change comes from the departure time of packets from the ATS. Because we select a transmission delay such that $TxD < I(1 - \frac{1}{s1})$, the departure time of $D^1_{j,k} \forall k \neq 1 \in \mathbb{N}$ from the ATS will be:

$$D^1_{1,k} \geq D^1_{1,1} + nI + n(k-2)I$$

The delay suffered through the IR by the first packet of the kth period, $\forall k \neq 1 \in \mathbb{N}$, of the first source is, when measured with $\mathcal{H}_{ATS} = \mathcal{H}_{GT}$:

$$
\begin{gathered}
D^1_{1,k} - A^1_{1,k} \geq D^1_{1,1} + nI(k-1) - \\
-A^1_{1,1} - nI/s1 - TxD - (n-1)\varepsilon - n(k-2)(I/s1+\varepsilon) = \\
= n(k-2)(I(1-1/s1)+\varepsilon) - TxD - (n-1)\varepsilon + nI(1-1/s1)
\end{gathered}
$$

We consider that no other packets exist in the network before $A^1_{j,1}$, so $D^2_{j,1} - A^1_{j,1} = 0$.

Case 3

• $TxD < \varepsilon$

This case is equal to the case considered with a FIFO system with infinitive service curve. Because the ε value is bigger than the transmission delay, the interval time between packets remain the same as in the original proof. The delay suffered through the IR by the first packet of the kth period, $\forall k \in \mathbb{N}$, of the first source is, when measured with $\mathcal{H}_{ATS} = \mathcal{H}_{GT}$:

$$D^1_{j,k} - A^1_{j,k} \geq (k-1)n\left(I\left(1 - \frac{1}{s_1}\right) - \varepsilon\right)$$

Bibliography

[1] *802.1AS-2011 - IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*. URL: https://ieeexplore.ieee.org/document/5741898.

[2] *802.1Qbv - Enhancements for Scheduled Traffic i.e Time-Aware Shaper*. URL: http://www.ieee802.org/1/pages/802.1bv.html.

[3] Matthew Andrews. "Instability of FIFO in the permanent sessions model at arbitrarily small network loads". In: *ACM Transactions on Algorithms* (Jul. 2009). URL: https://dl.acm.org/doi/pdf/10.1145/1541885.1541894.

[4] *Asynchronous Traffic Shaping drafts*. URL: http://grouper.ieee.org/groups/802/1/pages/802.1cr.html.

[5] Ehsan Mohammadpour Eleni Stai Maaz Mohiuddin Jean-Yves Le Boudec. "Latency and Backlog Bounds in Time-Sensitive Networking with Credit Based Shapers and Asynchronous Traffic Shaping". In: *2018 30th International Teletraffic Congress (ITC 30)* (Sept. 2018). URL: https://ieeexplore.ieee.org/document/8493026.

[6] Jean-Yves Le Boudec. "A Theory of Traffic Regulators for Deterministic Networks With Application to Interleaved Regulators". In: *IEEE/ACM Transactions on Networking (Volume: 26 , Issue: 6 , Dec. 2018)* (Nov. 2018). URL: https://ieeexplore.ieee.org/document/8519761.

[7] *Clock and ATS modules*. URL: https://github.com/guikoala/ns3-cpn.

[8] Anne Bouillard Marc Boyer Euriell Le Corronc. "Deterministic Network Calculus: From Theory to Practical Implementation". In: (Oct. 2018). URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9781119440284.

[9] G.810. *ITU : Definitions and terminology for synchronization networks*. 2004. URL: https://www.itu.int/rec/T-REC-G.810-199608-I/en.

[10] IEEE. "IEEE Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology—Random Instabilities". In: *IEEE Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology—Random Instabilities* (Feb. 2009).

[11] IEEE. "IIEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems". In: *IEEE Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology—Random Instabilities* (Jul. 2008).

[12] Arne Neumann Lukasz Wisniewski Rakash SivaSiva Ganesan Peter Rost Jürgen Jasperneite. "Towards integration of Industrial Ethernet with 5G mobile networks". In:

2018 14th IEEE International Workshop on Factory Communication Systems (WFCS) (Jun. 2018). URL: https://ieeexplore.ieee.org/abstract/document/8402373.

[13] Jinoo Joung. "Regulating Scheduler (RSC): A Novel Solution for IEEE 802.1 Time Sensitive Network (TSN)". In: *Electronics 2019* (Feb. 2019). URL: https://doi.org/10.3390/electronics8020189.

[14] Dong-Won Park Swaminatha Nataraja Arkady Kanevsky. "Fixed-priority scheduling of real-time systems using utilization bounds". In: *Journal of Systems and Software Volume 33, Issue 1, April 1996, Pages 57-63* (Feb. 1999). URL: https://doi.org/10.1016/0164-1212(95)00105-0.

[15] Tatsuya Maruyama Tsutomu Yamada Shouji Yoshida Mitsuyasu Kido Chikashi Komatsu. "NS-3 based IEEE 1588 synchronization simulator for multi-hop network". In: *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)* (Oct. 2015).

[16] Jin Seek ChoiBum Sik BaeHyeong Ho LeeHyeung Sub Lee. "Round-Robin Scheduling Algorithm with Multiple Distributed Windows". In: *International Conference on Information Networking ICOIN 2002: Information Networking: Wireless Communications Technologies and Network Applications* (Sep. 2002). URL: https://link.springer.com/chapter/10.1007/3-540-45801-8_76.

[17] *Linux Token Bucket Filter*. URL: https://linux.die.net/man/8/tc-tbf.

[18] *Linux Traffic Control*. URL: http://tldp.org/en/Traffic-Control-HOWTO/ar01s04.html.

[19] *LocalClock module PDF*. URL: https://github.com/guikoala/Clock-per-node-documentation.

[20] Jean-Yves Le Boudec Ludovic Thomas. "On Time Synchronization Issues in Time-Sensitive Networks with Regulators and Nonideal Clocks". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* (Jun. 2020). URL: https://dl.acm.org/doi/10.1145/3392145.

[21] *Massif visualizer tool*. URL: https://github.com/KDE/massif-visualizer.

[22] *Network Time Protocol Version 4: Protocol and Algorithms Specification*. URL: https://www.rfc-editor.org/info/rfc5905.

[23] *NS-3 a discrete-event network simulator for internet systems*. URL: https://www.nsnam.org.

[24] *NS-3 Real Time Simulator Manual*. URL: https://www.nsnam.org/docs/manual/html/realtime.html.

[25] *P802.1DG – TSN Profile for Automotive In-Vehicle Ethernet Communications*. URL: https://1.ieee802.org/tsn/802-1dg/.

[26] *Posix Linux clocks*. URL: https://linux.die.net/man/3/clock.

[27] Zhou Zifan Yan Ying Berger Michael Stübert Ruepp Sarah Renée. "Analysis and Modeling of Asynchronous TrafficShaping in Time Sensitive Networks". In: *Proceedings of 2018 14th IEEE International Workshop on Factory Communication Systems* (Jun. 2018). URL: https://doi.org/10.1109/WFCS.2018.8402376.

[28] *Repository with the code for the merge request*. URL: https://gitlab.com/nsnam/ns-3-dev/-/merge_requests/332.

[29] Matthieu Coudron Stefano Secci. *Per node clocks to simulate time desynchronization in networks*. URL: https://www.nsnam.org/workshops/wns3-2016/posters/per-node-clock-abstract.pdf.

[30] Mitsuyasu Kido Chikashi Komatsu Tatsuya Maruyama Tsutomu Yamada Shouji Yoshida. "NS-3 based IEEE 1588 synchronization simulator for multi-hop network". In: *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)* (2015). URL: https://ieeexplore.ieee.org/document/7324691.

[31] Jean-Yves Le Boudec Patric Thiran. "A Theory of Deterministic Queuing Systems for the Internet". In: (Version December 13, 2019). URL: https://ica1www.epfl.ch/PS_files/netCalBookv4.pdf.

[32] "Time-Sensitive Networking Profile for Industrial Automation". In: *IEC/IEEE 60802* (2019). URL: https://1.ieee802.org/tsn/iec-ieee-60802/.

[33] *TSN overview*. URL: http://grouper.ieee.org/groups/802/1/files/public/docs2017/tsn-farkas-intro-0517-v01.pdf.

[34] *Valdgrind tool for memory management and profiling*. URL: https://valgrind.org.