Time Sensitive Networking for Wi-Fi Based Wireless Industrial Environments

Presented by:
Arnold Baraka Doste Kinabo

Supervisor:
Dr Joyce Bertha Mwangama
Department of Electrical Engineering
University of Cape Town

Co-Supervisor:
Prof Albert A. Lysko
NextGen Enterprises and Institutions
Council for Scientific and Industrial Research

Submitted to the Department of Electrical Engineering, Faculty of Engineering & the Built Environment, at the University of Cape Town in partial fulfilment of the academic requirements for a Masters of Science in Electrical Engineering

20th August, 2021

Dedication

Hi Momz,

For always encouraging me to aim higher, even when I wondered if maybe it was more than what I deserved . . . *Kea leboha 'Mme Kinabo* :)

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.

2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.

3. This thesis is my own work.

4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signed by candidate

Arnold B. D. Kinabo
10th May, 2021

Acknowledgements

There are many whose support and efforts went into the writing of this thesis, and I'd like to take a few moments to acknowledge these.

I would first like to give thanks to my supervisor Dr Joyce Mwangama, who early on inspired me to make the switch to a more research-inclined degree, and to Professor Albert Lysko, whom she brought on as my co-supervisor when my thesis was first beginning to take shape. I could always count on their timely feedback and the expertise they freely lent; not only did it succour this research and bring my work to a higher level, it also resulted in a few scientific conference paper publications. In a similar way, I would thank Professor Thomas Magedanz for his unexpectedly enlightening lectures, more specifically on the topics of 5G and the Industrial Internet of Things. I would also recognise the students whose past theses were shared to me as templates for my own work. All this support is greatly appreciated.

To my colleagues at the Communications Research Group Lab, I say thank you. I found your critique to be constructive, and your advice quite helpful. I would also like to express my thanks to Ms. Nicole Moodley, the postgraduate administrative assistant, who usually enquired about my progress, and saw to it that I was settling into the department well.

Finally, I wish to convey my utmost gratitude to my friends and family. To Fr John and my family at Kolbe Chaplaincy, you were by far the best part of my UCT experience. With you, my spirit grew to match whatever trials college life (and the outside world) threw at me. I would extend a special thank you to Perose, for all the times we studied and worked together; those nights at Campus Key will remain fond memories. To Amanda, I am deeply grateful for all the support in the many forms it came: the prayers, the motivating words, and the unwavering confidence you had in me. Your company was something to be cherished in the current world crisis. And to all my other friends who provided welcome distractions to settle my mind outside of research, you mean the world to me. In ending, to my parents, Doste and Tina, I want to thank you for investing in my education, be it financially, emotionally, and the many untold sacrifices you made out of love for me. Along with my siblings, Herieth and Eric, your invaluable support will forever be treasured. In all things, I see you, Yahweh; for the triumphs and the pitfalls, thank you for it all.

Abstract

In production industries, mission-critical assignments require networks characterised by deterministic low latency, dedicated bandwidth resources, and, chiefly, reliability. Several fieldbus technologies are specially placed for this. Their commonality is that they run on standard Ethernet. The relatively new Time Sensitive Networking (TSN) is among these technologies. It is a set of Ethernet standards that guarantees determinism for real-time use-cases. TSN sets itself apart in that it is vendor-agnostic. And so, it promotes interoperability among standard-conformant devices. Being based on Ethernet, even TSN is plagued by downsides associated with cabled networks, most importantly, the limited range and mobility. In this regard, wireless networks are an attractive option – it would be an opportunistic venture to operate TSN in the wireless medium. Previous works have tried to address how this can be done, but as yet, it is an open problem. The issue is that most wireless networks are not optimised for determinism. Most lack the scheduling, synchronisation and other capabilities that timing-stringent applications require. Wi-Fi, for instance, suffers from many issues stemming from randomised medium access and interference, which remove the predictability from its communications. Critical TSN traffic needs special consideration when run with other services in current Wi-Fi. That being said, the key research question is: can one contend with the problem of transmitting TSN and non-TSN traffic together in the same wireless network?

To answer this, the work develops a TSN simulation model that operates in Wi-Fi, whose test results can be studied to aid in analysing wireless TSN. The prototype model runs in a simulation environment, and was developed using methods that involved reusing and modifying the present wireless architecture to support the TSN traffic. Through the course of several iterative experiments, it was revealed that although the current generation of Wi-Fi can support TSN traffic, it does so inefficiently. Even with no interference, the TSN traffic experiences low losses only when the network capacity utilisation is very low, below a small percentage value. Considering the typically low demands on bandwidth in many TSN applications, this inefficient operation may still be sufficient for operating TSN over existing Wi-Fi networks. For more robust and general applications, Wi-Fi requires further enhancements to its mode of operation in order to support prioritisation of TSN traffic and more accurately cope with higher loads.

Table of Contents

List of Figures

List of Tables

List of Acronyms

3GPP	3rd Generation Partnership Project
4IR	4th Industrial Revolution
5G	5th Generation of Mobile Communications
5GS	5G System
ACK	Acknowledgement
AP	Access Point
API	Application Programming Interface
BMCA	Best Master Clock Algorithm
BSS	Base Station Subsystem
CBS	Credit-Based Shaper
CNC	Central Network Controller
CUC	Centralised User Configuration
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Arrays
FRER	Frame Replication and Elimination for Reliability
GCL	Gate Control List
GUI	Graphical User Interface
IDE	Integrated Development Environment
IE	Industrial Ethernet
IETF	Internet Engineering Task Force
IoT	Internet of Things
IIoT	Industrial IoT
IRT	Isochronous Real-Time
ISM	Industrial Scientific Medical
LAN	Local Area Network
LTE	Long Term Evolution
LTE LAA	LTE License-Assisted Access
LTE-U	LTE-Unlicensed
LTS	Long-Term Support
MAC	Media Access Control
NED	NEtwork Description
NeSTiNg	Network Simulator for Time-Sensitive Networking
OS	Operating System
OSI	Open Systems Interconnection
PCP	Priority Code Point
PLC	Programmable Logic Controller
PTP	Precision Time Protocol
RAM	Random Access Memory
RTA	Real-Time Appplications
SNIR	Signal-to-Noise-Plus-Interference Ratio

SSID	Service Set IDentifier
TAS	Time-Aware Shaper
TCP	Transmission Control Protocol
TSN	Time Sensitive Networking
UDP	User Datagram Protocol
UE	User Equipment
ULL	Ultra-Low Latency
URLLC	Ultra-Reliable Low-Latency Communication
VLAN	Virtual LAN
VM	Virtual Machine
WSL	Windows Subsystem for Linux
WSN	Wireless Sensor Networks
XML	eXtensible Markup Language

1. Introduction

1.1 Background

Internet of Things (IoT) is gaining momentum and becoming a critical component of everyday lives, with the global market spending predicted to spill over the trillion mark, reaching $1.2T in 2022, while it was $235B in 2017 [1].

The Internet of Things (IoT) is a cornerstone technology propelling the next generation of communication networks. In a world where more and more smart devices are produced that are able to connect and interact with each other, the industrial sector is one notable use case seeking to take advantage of these capabilities. The devices participating in an IoT communication need some level of coordination in the way events are scheduled to occur. For example, in a vehicle manufacturing plant like that in Fig 1 [2] which is in process of assembling cars, having one of the parts put in at the wrong time would create chaos in the production line, whose effect is likely to get multiplied if prolonged. Thus, reliable time synchronisation is of utmost concern in the controlled setting of the factory floor.

Fig 1: Industrial IoT for Factory Floor Automation

Time Sensitive Networking (TSN) is the IEEE 802.1Q standard that seeks to provide deterministic messaging over Ethernet, specifically on the physical and data link layers of the network [3]. It envelops a set of standards developed by the TSN task group under the IEEE 802.1 working group. TSN relies on strict synchronisation between communicating entities in a network. It has gained usefulness in industrial networking and factory environments. Scheduling and traffic shaping need to be performed on all network devices, as well as communication path management and selection mechanisms, infused with reusable Software Defined Networking (SDN) concepts to cater for evolving topologies [4].

In particular, TSN is useful to enable IT equipment with a shared concept of time so they may be utilised as industrial-grade devices. It allows for these standard networking tools to converge with industry technology; with TSN, the resulting network begets determinism and low latency guarantees [5]. It is a technology whose focus is time, and delivering information in a predictable amount of time. The writer believes predictability directly correlates to increased efficiency because, in [6], a predictable behaviour is a requisite for deterministic networking, of which TSN establishes this predictability through its standards [6].

TSN currently comprises a list of standards that make up the Ethernet standard 802.1Q. It does not have an equivalent wireless standard as of yet. The key to understanding TSN lies in understanding how these standards work. The main ones as maintained by the IEEE Standards Association in [7]–[10] are described thus.

1.1.1 Major Standards – Synchronisation

TSN firstly supports the synchronised relay of information. This is important because, in order to have a deterministic communication, connected devices need to have a shared concept of time [5], that is, they should have the same understanding of time. The devices should be in sync so that certain operations execute concurrently. If neglected, it would cause machines to receive instructions at different times and carry them out in a mismatched order, potentially setting off many dangers. **IEEE 802.1AS** [7] allows for this. 802.1AS is the predecessor to P802.1AS-Rev [8]. It handles timing and synchronization between TSN devices existing in networks that consist of full-duplex IEEE 802.3 (Ethernet) and 802.11 (Wi-Fi) LANs (Local Area Networks) [11]. It is based on, and includes a profile of IEEE Std 1588-2008 [11], also termed IEEE 1588v2 PTP (Precision Time Protocol), which is a protocol for synchronising clocks in a network's nodes. 802.1AS is therefore a profile of 1588v2 or PTP that provides a subset of the PTP features like "performance specifications for switches as Time Aware Bridges" [12].

TSN works similar to IEEE 1588's hardware-timed synchronisation in that a grandmaster is selected from among the time-aware systems, using the "best master clock algorithm" (BMCA)[1]. As defined in the standards in [7], a time-aware system is a bridge or node that can communicate synchronised time received on one port to other ports using the 802.1AS protocol. This grandmaster is the time-aware system determined to have the best clock [7], whose job it is to propagate timing packets among the network elements by forwarding to directly connected time-aware systems, which, in this situation, are termed as slaves. These elements or time-aware systems have to adjust the synchronised time which they receive. They do this to account for the delay in transit from when they left the grandmaster to when

[1] The BMCA compares among several attributes, to determine which device should be the Grand Master: Priority 1 (which may be chosen based on proximity to the network, or how likely a device is to be removed versus those that are fixed), Class (the role of device), Clock Accuracy (the time source accuracy), offsetScaledLogVariance (or clock variance), and others, ultimately ending with the port number and MAC address of the clock as the final decider attribute. This attribute, offsetScaledLogVariance, characterises precision and frequency stability.

the packet arrived. They add this propagation time to the synchronised time and if the device is a time-aware bridge it then forwards this corrected time along with all other delays it may encounter in forwarding to the other time-aware systems it is attached to. In this way, all the time-aware systems sync their system clocks to that of the grandmaster. For this to work, [7] has identified that these two time intervals must be precisely known, that is, the forwarding time (known as residence time) and the transit or propagation time.

There is a difference between the two types of synchronisation: hardware and software. In hardware synchronisation, the timing packets can achieve an accuracy within the order of a microsecond, as opposed to if it were done via software synchronisation, which would reduce the accuracy to within a millisecond. Where a TSN network differs from one employing a typical 1588 protocol (please see following sub-section on 1588) is that, firstly, TSN will generate an error should the synchronisation of its devices fall out of certain bounds. Secondly, its timing packets receive priority scheduling [13]; they do not wait for other transmissions in the buffer as much as regular frames. So, because they do not wait, no matter the amount of traffic the network must contend with, all device clocks should always be tightly synchronised. Still, it is important to understand the original IEEE 1588 timing mechanism because of the bearing similarities to TSN, which the next section shall now get into.

Mechanism of IEEE 1588 Precision Time Protocol

IEEE 1588 enables the synchronisation of clocks to the order of hundreds of nanoseconds, thus enabling precise synchronisation among network devices used in measurement and control systems, factory automation and other types of network communication. It fixes the time difference that exists between the master, or grandmaster clock and slave clocks which are used by the devices in these communication systems. Cho et al. [14] attributed the clock skew to two primary sources of error: the clock offset, and the message transmission delay. The protocol works to ensure that all nodes in the network use the same timer values. For this task it uses two sets of messages (the details of which were reviewed in [14]):

1st Set of Messages

It is initiated by the master clock, and it is used to correct the clock offset. Two messages are sent: *sync* and *follow_up*. They arrive in that order. The slave uses its local clock to timestamp the arrival of the sync message. The second message, the *follow_up* message, has been timestamped by the master clock to indicate the transmission time of the prior sync message. The sum of the offset of the slave and the message transmission delay (the two key sources of error in timing) is the difference between these two timestamps.

2nd Set of Messages

The objective of the second set of messages is to account for variations in network delays. The slave clock initiates this phase by sending a delay request message to the master clock, timestamping it as it is sent out. This delay request is timestamped upon arrival at the master. Feedback is sent to the slave through a delay response message, bearing the arrival timestamp issued by the master. The slave calculates the difference between the two timestamps as the slave-to-master delay value. The average of the two delays is important; it is what the slave uses to adjust its clock so that both master and slave are synchronised.

In this way, offset correction and message delay correction is effected. It occurs periodically because these two clocks tend to drift apart over the course of time.

As per [15], it should be noted that the discussed IEEE 1588v2 PTP mechanism is used for synchronisation in packet networks. SyncE can also be used. Other kinds of networks would require different schemes: GPS/GLONASS for satellite networks [15]; similar GNSS (Global Navigation Satellite Systems) systems for base stations in LTE; though it is believed that mobile networks, especially future networks including 5G, are best served by a combination of highly accurate network-based and satellite-based synchronisation methods [16].

1.1.2 Other Major Standards for Scheduling and Frame Pre-emption

At the heart of TSN communication is the scheduling of traffic in queues in order to provide the sought-after deterministic messaging. This scheduling operation is governed by the "time-aware shaper" (TAS) principle [17] that is standardised as **802.1Qbv** [9]. With the introduction of these queues, messages will only be allowed to transmit during their scheduled time windows [17]. Transmission gates that are in an "open" state will be allowed to execute their queues, while those in a "closed" state will have to wait their turn. Because of this, there will be no danger of interrupting critical message queues in their transmission. Thus, the TAS can guarantee a maximum bounded latency for the network [17] and herein lies the property of determinism.

But what of those critical messages whose gates are in a closed state whose urgency surpasses those non-critical queues which are in an open state? These critical messages would otherwise get delayed because they do not have the right of transmission at that point in time. TAS guarantees the upper limit of latency in a network, ensuring that critical messages will always be immune to interruption in their transmission, but it sets no minimal latency for the network. That is where **802.1Qbu** [10] comes in with its frame pre-emption mechanism. Essentially, for optimum network utilisation, there needs to be accounting for the delivery of critical messages that take precedence over non-critical messages, even though the channel is currently still being used to transmit the non-critical frames. With 802.1Qbu in effect, a link with frames of lower priority can be interrupted mid-way to transmit higher-priority Ethernet frames, all without losing the original messages [18]. In fact, upon completion of the critical

frames, the first message frame can continue its transmission from the exact point it left off [17].

There are other standards that promote the goal for reliability, such as **802.1CB** [19] – Frame Replication and Elimination for Reliability (FRER), and **802.1Qcc** [20] – Stream Reservation Protocol (SRP) Enhancements and Performance Improvements. These will not be discussed at length in this document.

With these major standards in play, the mode of operation for TSN is as illustrated in Fig 2.

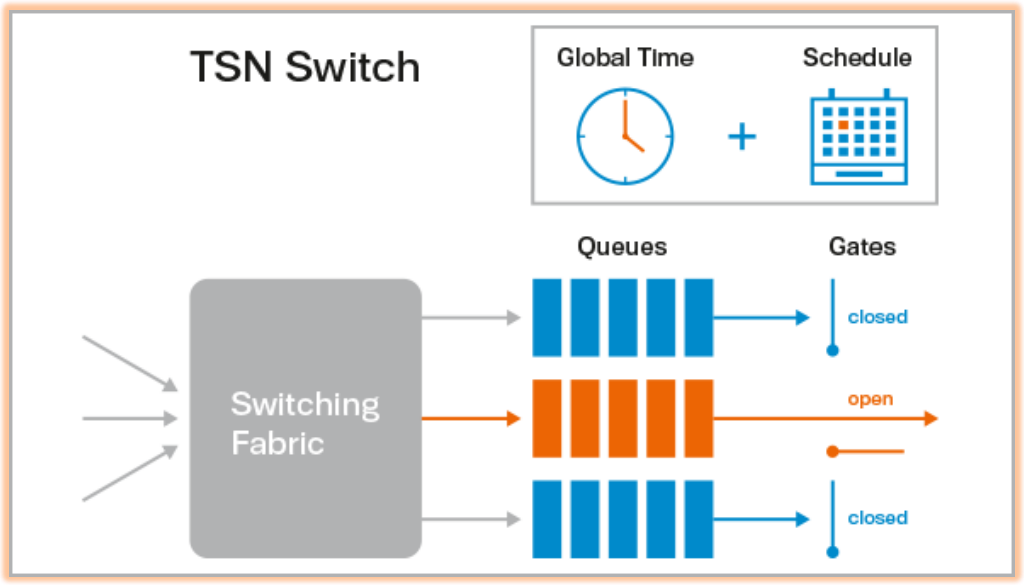

Fig 2: TSN Switch Operation [21, Fig. 1]

The switching fabric represents any TSN-capable switches in the network. On closer inspection of these, one finds the messages of different priorities in their different queues. These messages may belong to various services running on the same physical connection. Recounting on what was earlier mentioned, the message queues will be released depending on the states of their gates, whether open to transmit, or closed to wait their turn. All these switching devices are governed by a global time that they are all synchronised to, as well as a scheduler that defines when a specific message queue is supposed to transmit.

In all prior art found so far, TSN has been employed in a largely wired medium. Even in simulation environments, authors have opted to try and capture as many of its features as they can, to produce as complete a TSN model as possible, but still over a virtual Ethernet connection. They have done this with success to varying degrees; some have focused on simulating two standards through use of their own source code they developed [22], while others have developed a framework specifically for simulating TSN, which provides simulation models to separately showcase different TSN mechanisms [23]. Even so, little work has been

done to roadmap the setting up of TSN over a wireless connection [24], [25]. This work will look further into this wireless capability in its next chapters. A brief review contrasting wired and wireless media now follows.

1.1.3 Wired and Wireless Media

Reliability is the first aspect that degrades in a wireless medium as opposed to a wireline one, simply because collisions cannot be mitigated as well, owing to the greater number of users. This causes congestion, which is the main reason for packet loss and longer latencies in networks [26]. Considering the case of TSN in industry, reliability is non-negotiable. A message should be transmitted within its scheduled window. Interference from other sources is a major point of concern here. Wireless LAN signals are susceptible to interference from a range of other devices, though, in a contained plant-floor, this interference can be controlled. If the interference is from other parties, that is, from outside the factory, oftentimes the plant is large such that the distance they cover is great enough for the interference to decay. If on the other hand the interference source is in-plant communications, other methods can be used to mitigate or solve the disruption. Interference from multipath effects and electrical disturbances can be eliminated by methods such as OFDM (Orthogonal Frequency Division Multiplexing) and MIMO (Multiple-Input Multiple-Output) for the former, and galvanic isolation for the latter [27]. In any case, because of higher packet loss, packets have to be retransmitted more often in wireless media [24]. This accounts for longer latency times since, for one, it extends the PPDU[2] duration, or packet duration. Although, for many industry applications, this extended latency may not be the biggest problem; reliability is the non-negotiable factor, even at the cost of slightly longer latencies. Due to its proneness to dropping packets, a standard wireless channel isn't as reliable as a wired one.

It is also easier to achieve higher bandwidth in wireline networks. Even though this is so, it does not have much foothold in the case of factory communications because there are usually no strong expectations for high-capacity transmissions for IoT applications. Machines generally do not exchange so much data, except for the slowly rising case of real-time remote control of fast-moving machinery requiring instant control feedback [28]. With this reasoning, standard capacity rates hold well, unless the network somehow handles so much data it becomes congested.

Another factor, though not likely to affect the delivery of packets and timing directly, is security. Environments with dedicated wired links are generally more secure than wireless channels that are "open" for all sorts of transmissions *and* interceptions. To address security concerns, wireless security certification programs such as WPA (Wi-Fi Protected Access) 2 and,

[2] PPDU is an abbreviation inside an abbreviation for PLCP (Physical Layer Convergence Protocol) Protocol Data Unit. It is what's used to measure a unit of information that is transmitted among peers in a packet-switched data network. Some texts may refer to it as simply the Physical Layer PDU, or more painless still, the packet duration.

more recently, 3 were developed for encryption in wireless channels. This upgrade from the older Wired Equivalent Privacy (WEP) was aimed at making them more secure like wired networks.

Other points of difference include: the expenses involved in setting up and maintaining these different networks, with wired channels predominantly being costlier; the increased mobility with wireless devices, and; the increased range that hard-wiring allows, with no compromise in signal strength.

This work recognises those features of wireless media that would be beneficial to have in the wired case. It thus aims to develop a TSN simulation model that can be reused in wireless environments requiring strict timing like mission-critical IoT applications. The writer believes there are economic savings the industrial world stands to benefit from averting potential disasters caused by poorly synchronised or scheduled operations, since, as posited in [5], synchronisation is critical for industrial manufacturing efficiency and product quality.

By investigating wireless TSN, this work also has the potential to foster increased coordination in machine-type communications (MTC), allowing them to take a fuller part in the 4th Industrial Revolution (4IR). This work is relevant as it aligns with the South African Department of Science & Technology's priority of driving the nation towards a knowledge-based economy. Knowledge-based industries backed by innovation are what generate wealth for many of the world's strongest economies [29].

1.2 Problem Statement

IoT data exchange requires steady network connectivity to facilitate interactions between intelligent devices. In the next generation of communication networks, these limits are pushed further to realise ultra-reliable low-latency transmission [3]. More specifically, in the industrial setting, the widely employed Time Sensitive Network (TSN) standard offers this aspired level of reliability in wired media. Latency can suffer somewhat, but the focus is placed on guaranteeing a reliable flow of data above all [24].

As IoT continues to grow, more use cases make use of the wireless medium, for reasons such as mobility, as discussed priorly. The challenge lies in the fact that TSN relies on a physical link for communication. Few authors have tried realising it in any medium other than standard Ethernet [24], [25]. Achieving deterministic communication over wireless infrastructure will require a review to determine what factors stand to impede the reliability. This forms the basis of the problems that this work seeks to investigate.

1.3 Motivation and Research Gap

This sub-section begins by briefly looking into Ethernet to see what motivated the development of TSN, before it reveals the research gap in TSN communications.

1.3.1 Factors that affect Determinism in standard Ethernet infrastructure

The factors that affect determinism in Ethernet include those that target the synchronisation and scheduling capabilities of a network. Why can't networked devices even in a standard Ethernet infrastructure efficiently relay information for a communication that is deterministic? The answer to this can be broken down into two main parts, that is, a problem in scheduling and a problem in synchronisation.

1.3.1.1 The Scheduling Problem

In a standard Ethernet network, all node devices in a channel of communication essentially have to always be listening in order to know when the channel is free for them to transmit. Simultaneous transmissions from devices in the same "collision domain" or segment result in collisions that render the communication failed. To aid with this, Carrier Sense Multiple Access with collision detection (CSMA/CD) and with collision avoidance (CSMA/CA) were among the earlier methods adopted for Ethernet and Wireless networks, respectively [30], [31]. Speaking on the case of an Ethernet infrastructure, this method is inefficient, which is why switches began to be used instead to eliminate collisions. They can create duplex links between themselves and other devices, thereby eliminating any chance of collisions with other communicating devices. This ensures a higher chance that frames are forwarded and received. Switches alone, however, fail to effect a truly deterministic communication since, as multiple message flows are being handled at once, for time-sensitive data, there is no predicting when it will arrive at its destination. This lack of predictability translates into, firstly, the increased possibility of longer latency times among the network devices, and secondly, scheduling cannot be performed for real-time deterministic communication when delivery times are unpredictable.

1.3.1.2 The Synchronisation Problem

It has already been stated why synchronisation is an important challenge in time sensitive communications. Ethernet is asynchronous in nature. This becomes a problem on the plant-floor when networked devices connected on standard Ethernet need to execute certain actions in sync. Having a common time among networked components is of paramount importance in guaranteeing that instructions get carried out at the right time and in the proper order. In order to ensure that all networked devices have a shared concept of time, standards, such as Synchronous Ethernet (SyncE) by ITU-T and IEEE's 802.1 AS-Rev and IEEE 1588 Precision Time Protocol (PTP), are in use.

Collectively, these two issues - scheduling and synchronisation – must be taken into due consideration. Meeting the requisites for these issues will speak a lot on minimising latency, reducing packet delay variation (jitter) and packet loss [22], increasing efficiency in resource management, and, most importantly, the overall reliability of the network [6], which is an essential property for realising many industrial communications [6]. TSN is intended to take care of determinism – scheduling and synchronisation – for the case of wireline infrastructure.

1.3.2 Research Gap

As mentioned before, all TSN solutions proposed by various bodies aim, through and through, to support its implementation over an Ethernet connection. The opportunistic case of TSN communications over a wireless medium is left largely unaddressed, as has been evidenced in [24], [32]–[34]. What this means is that synchronisation and scheduling, and determinism in general, are still open problems as pertains to wireless communications.

1.4 Research Questions

This work seeks to address the following questions:

- How do wired and wireless channels differ in the context of industrial communications?
- What are the factors affecting reliable transmission in wireless TSN?
- Can one contend with the problem of transmitting TSN and non-TSN frames together in the same wireless network?
- Can one translate/replace any of the components in the existing wired TSN already deployed in the industry into a wireless design? How would this configuration look like? Would there be any differences in performance?

1.5 Objectives

This study is aimed at investigating the issues that hinder the realisation of TSN over a wireless network.

In order to achieve the aim, the following objectives guided this work:

- To understand the differences between wired and wireless approaches as regards industrial communications
- To consider the factors that affect reliable transmission in wireless TSN
- To develop a TSN simulation model to cater for IoT systems
- To use the model to study and highlight the features that TSN needs to be reusable in wireless mission-critical environments.

1.6 Research Outputs

These are the publications that were developed during the registered period of Masters work:

1. A. B. D. Kinabo, J. B. Mwangama, and A. A. Lysko, "An Evaluation of Broadband Technologies from an Industrial Time Sensitive Networking Perspective," 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS'21), Coimbatore, Tamil Nadu, India, 19-20 March 2021, 8pp. (**Published**) ISSN 2575-7288, and ISBN 978-1-6654-0521-8, 978-1-6654-0519-5, and 978-1-6654-0520-1.

2. A. B. D. Kinabo, J. B. Mwangama, and A. A. Lysko, "An Overview of Time-Sensitive Communications for the Factory Floor," 16th IST Africa 2021, Uganda, 10-14 May 2021, 9p. (**Published**)
3. A. B. D. Kinabo, J. B. Mwangama, and A. A. Lysko, "Wi-Fi Time Sensitive Networking," SATNAC 2021, South Africa, 29 August – 1 September, 6p. (**Submitted**)
4. A. B. D. Kinabo, J. B. Mwangama, and A. A. Lysko, "Wired or Wireless for Time-Sensitive Applications in Industry?" IEEE Access Journal. (**In preparation** – 29p.)

The source code used in the development of the proposed solution is hosted on Github, accessible at: https://github.com/ArnoKinabo/NeSTiNg-Wireless-TSN.

1.7 Scope and Delimitations of the Study

This study aims to develop a wireless TSN simulation model based on Wi-Fi to cater for IoT systems with stringent timing requirements, although the system will aim for reliability before other factors such as low latency.

Having stated this, a key limitation to this work is that it has had to incorporate simulation techniques to address the wireless TSN solution. Ideally, the writer would have liked to develop the solution in an environment where it could be tested with industry-standard equipment. This was not possible, however, because of budgetary constraints. Instead, the writer utilised simulation programs for the solution.

Furthermore, the scope is limited to developing the wireless TSN model based on the present standards and available software APIs (Application Programming Interface). It leverages the state-of-the-art technologies and standards in their present state to develop the solution.

As with any communication or process that involves data transfer, there is always an aspect of security involved. This work acknowledges this and purposely leaves it out: Adding this aspect would make the scope too broad to effectively focus on. The solution proposed in this work thus excludes information security such as those areas pertaining to cryptography, authentication, confidentiality, and the integrity of user data.

A final key delimitation of the research should be defined: the situation addressed is typically that of plant-floor environments where the network traffic is centrally managed, that is, it is not carried through Wide Area Networks (WANs) such as the open internet, or in situations that require IP (layer 3) routing. In such circumstances, it is easier to propagate traffic for a deterministic communication. Hence, the TSN LAN described herein excludes any such third-party networks.

1.8 Thesis Organisation

The remainder of this work is organised as follows: Section 2 details the current status of industrial communications, including fieldbus technologies that are used to effect

determinism in factories, a review on TSN as well as research efforts to pursue it in the wireless direction, and a possible alternative to TSN, DetNet [26], which caters to Ultra Low Latency (ULL) networks; Section 3 outlines the requirements of the framework tool, and the steps considered in its design, wherein reasoning behind the chosen methodology is also discussed; Section 4 offers a description of the implementation tools, and the procedure for implementing the prototype wireless design, run in a discrete event simulation environment in which it will be tested against the wired implementation; Section 5 presents various tests and their results to benchmark the model's performance against the wired alternative, along with an analysis to evaluate the wireless TSN method; and lastly, Section 6 concludes this work, illustrating the significance of the study, and providing insight on research areas for further work.

2. A Review of the Literature

TSN was designed to fulfil different communication requirements in the industry: automation, automotive, audio, video, and more [31], [35]. In delving into the topic of industrial automation, as that is where TSN is most applied, a subset of sources has been chosen based on their relevance to the following questions:

1) What standards and technologies are presently available for real-time communications enabling automation on the factory floor?
2) How do these technologies compare to each other? How does TSN fit in?
3) What is the current state of TSN?
4) How is it employed by different technology giants in the industry?

The emergence of IoT in the industry is just one of many possible use cases of the Internet of Things, however, it is also one of the fastest-growing verticals. Columbus [1], in his market forecast, showed that figures reported by IoT analytics see the market for the Industry 4.0[3] products and services alone growing from $119B in 2020 to $310B in 2023. This is quite a substantial figure given that it was only in 2017 when the combined markets of the Internet of Things, and not just its industrial share, were sitting at a total of $235B [1].

Because industrial IoT (IIoT) is so disruptive, scientific bodies and tech companies alike have voiced their opinions on the subject matter. Many sources detail how they have applied, or foresee applying deterministic wired and wireless communications technologies in their own environments. This review of the literature will present their findings with reference to the previous four questions. A similar overview of these time-sensitive communications for the factory floor is awaiting to be published in the IST-Africa 2021 Conference, the 16th in an annual series.

Question 1: What technologies are presently available for real-time communications enabling automation in industry?

2.1 Review of Industry Standards and Protocols

There are quite a few network technologies built over Ethernet that are used in the manufacturing and process automation industries to guarantee deterministic delays in communication. In answering the first question, a review of the current chief fieldbus communication technologies is conducted. A thorough examination of said technologies is dealt with most appropriately in the sources [36]–[45], and they are discussed below. It is worth noting that major technologies such as those under review are all built on open standards. Aside from these major ones, many other technologies aren't open, since either

[3] Industrie 4.0 or Industry 4.0 is the subspace of the 4th Industrial Revolution that pertains to industries.

the aspects of their implementation require use of proprietary devices from the vendor, or their source code is not freely available.

2.1.1 PROFINET

Process Field Network, or PROFINET, is a "100% Switched Ethernet" [36] protocol that complies with IEEE 802.3 and which seeks to address factory automation. It is an open standard that was developed and is maintained by the PROFINET International group in [36]. PROFINET can be scaled to meet different functional requirements by the combination of Conformance Classes, which build upon one another in increasing layers to extend each other's functionality, as seen in Fig 3 below redrawn from the source in [36, Fig. 1]. In line with these classes, PROFINET comes in three versions: Versions 1, 2 and 3.

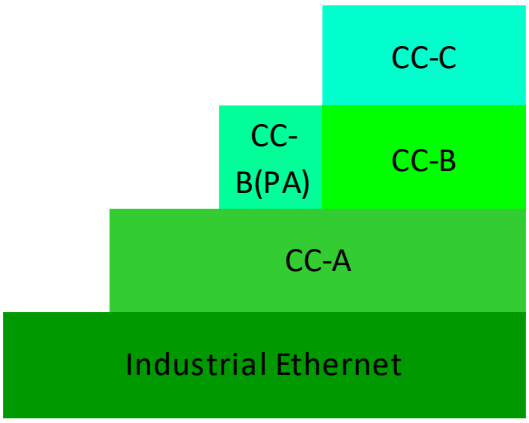

Fig 3: Conformance Classes of PROFINET

Version 1 is based on Conformance Class A (CC-A) and, according to [37], it provides component-based automation. It is not time-sensitive because, according to Rostan [37], it is built on best-effort messaging principles. On the other hand, versions 2 and 3 are the main approaches since they are more time-aware. Version 2, based on CC-B, engulfs a "soft" real-time[4] (RT) approach [37] and is thus known as PROFINET RT. It allows the transmission of high-priority Ethernet frames through channels that have VLAN (Virtual LAN) prioritisation schemes in place [43]. According to AMMC (Applied Machine & Motion Control) Golden Motion analytics [43], with PROFINET RT, systems can hope to experience cycle times (response time) of 1-10 milliseconds; hence it is best suited for those I/O applications that require precise digital or analogue I/O control. In comparison, version 3, based on CC-C, is an isochronous real-time (IRT) approach [37]. It employs hardware-synchronised switches to cater for applications that work on even shorter cycle times: below one millisecond according to [43]; examples include motion control applications. Another important feature of this

[4] A hard real-time system is very restrictive; it has higher performance metrics and shorter delays. If an operation misses just one deadline, this corresponds to a system failure. In contrast, a soft real-time system has a higher tolerance, and it can miss more than one deadline. Although, if this happens too often it will degrade the overall performance of the system.

PROFINET IRT, as documented by Hibbard [45], is in its ability to reserve bandwidth. This bandwidth can be used to transmit critical data frames immediately and reliably as needed [43].

PROFINET RT and IRT are collectively called PROFINET I/O [37].

2.1.2 EtherNet/IP

The IP in the name stands for Industrial Protocol. EtherNet/IP is an industrial network solution for providing manufacturing and process automation by employing the Common Industrial Protocol (CIP) [46]. CIP is so named because it provides both a common data organisation and a common messaging for control systems in industry, as noted in other unpublished sources [47] affiliated with [46]. It utilises both connection-oriented and connectionless protocols, that is, TCP/IP (Transmission Control Protocol/Internet Protocol) and UDP/IP (User Datagram Protocol/Internet Protocol) respectively, to transfer data. And this it does through various communication mechanisms: event triggers, multicast, and point-to-point connections to name a few [43]. The main distinguishing feature of EtherNet/IP lies in how it encapsulates its data in the transport layer. [43] noted how it is able to distinguish between two kinds of messaging: implicit (data only) and explicit (requests and denies); the implicit I/O messages, it sends using UDP, while the explicit messages are sent by TCP/IP. Under the technologies compared in this section it is the only real-time technology that is based completely on Ethernet standards [45].

In as far as machine-to-machine communications are concerned: CIP adapts the CIP Motion architecture discussed in [44], which allows for a peer-to-peer connection to be set up through which a "producing" controller can propagate high-speed motion data to "consuming" controllers or devices via a single multicast connection [44]. The use cases documented by Zuponcic [44] saw the producing controller sharing axis information to coordinate the other devices such as motors. These devices receive a common reference, and, in this way, CIP Motion allows for synchronising and coordinating the speed at which devices operate. Zuponcic described how it might also similarly be used in robotics control on the factory floor, among other things.

[46] highlights the advantages of EtherNet/IP over other technologies like PROFINET. The first and chief of these is how it is the only real-time technology built solely on Ethernet standards, so it uses all the transport and control protocols of traditional Ethernet such as TCP and IP, and media access and signalling technologies from common off-the-shelf Ethernet interface cards. It also requires no special hardware because it is simply an application layer protocol sitting on standard Ethernet and TCP/IP. Another advantage is how a range of unique devices on different networks can be accessed and configured from a single access point using one common mechanism (no need for vendor-specific software). Its main disadvantage in comparison to other standards is its limited bandwidth [45] and performance range [37], [45].

Speaking to these limits, the group from [46] report in other non-academic sources how EtherNet/IP PLCs (programmable logic controllers) are only capable of transferring 500-byte sized packets.

2.1.3 Sercos III

As a technology that follows the master-slave structure, Sercos III provides deterministic communication by allowing for the devices that employ this protocol to be very precisely synchronised: according to [39], up to 100 devices can have a Time Error of 1 microsecond. Sercos III is the third generation of the Sercos Interface, according to the founding company. It is a portmanteau for Serial Realtime Communication System. The technology supports both cyclic and acyclic communication, that is, deterministic or scheduled, and best-effort communication. Nsaibi, Leurs and Schotten [39] explained that the master node would send "cyclically real-time data" to its slave I/O components by incorporating summation frames. A summation frame will contain the data of several devices in it in a sort of accumulating or "summing" effect. The summation frame principle can only be used, however, when the network node topology is a daisy chain (a single ring) or closed ring [43]. Once the frame size is at its maximum, then any new information is appended to another frame [39].

Different sources have classified frames according to different criteria, although examining them reveals that they fit into one another. In [37], there are two classifications of data frames: a frame for input data and another for output data. According to Szancer et al. [41], the master node issues two frames, classified as: Master Data Telegrams (MDT) and Acknowledge Telegrams (AT). A configuration of these telegrams in the Sercos communication cycle is shown in Fig 4 [48].

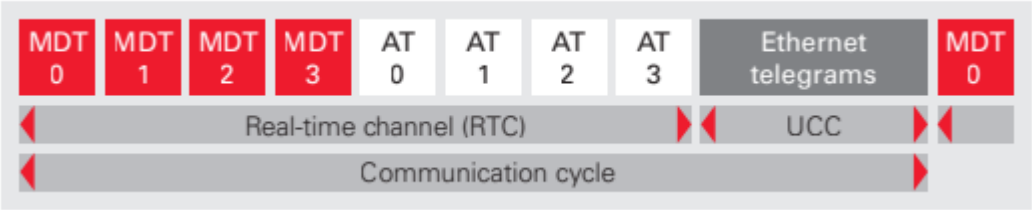

Fig 4: Configuration of the Sercos Communication Cycle [48]

Details on these telegram types in [41] reveal that MDTs help the master relay information to its slaves and are thus filled by the master and read by the slaves; whereas, ATs are a way for the slaves to provide responses back to the master and to communicate with other slaves, and so they are filled up by the slaves. The Sercos organisation [48] maintains that each slave will populate its assigned device channel in the AT before passing it on to the next slave device. And, in line with the summation frame principle, if the AT reaches its maximum size, then another AT will be used for the next device. The same goes for MDTs. So, in this way, the MDTs being issued out to slaves can be regarded as input, and the ATs that are fed back to

the master from the slaves as output, which supports the classification by Rostan [37]. Another key feature of Sercos as documented by Hibbard [45] is that each real-time telegram is processed twice per cycle: a telegram as it passes through a slave node will be processed once as it leaves that node, and once as it returns on its way back. This characteristic allows for devices to execute communication amongst each other in a single cycle without having to route their data back to the master [45]. Hibbard also noted how Sercos is another real-time Ethernet standard that reserves bandwidth, doing so for the isochronous or real-time channel, and asynchronous (IP channel) data traffic.

2.1.4 EtherCAT

Ethernet for Control Automation Technology (EtherCAT) is another master-slave arrangement that caters for hard and soft real-time computing. It shares all but a few of the features of Sercos III. As in Sercos, frames are still "processed on the fly" by slaves [37], that is, they are processed without stopping. But in this case, with the use of special slave controllers that can be implemented in different options: as Field Programmable Gate Arrays (FPGAs), Application Specific Integrated Circuits (ASICs), or as microcontrollers [37]. From Wang et al. [38], the master node is connected to its slaves in a logical ring topology over standard Ethernet cable. It is as well a summation frames network, so each frame will be processed from node to node in sequence until it reaches the end slave, after which it turns back.

The mode of operation of EtherCAT allows for cycle times to be drastically reduced, according to Hibbard's analysis in [45], leaving EtherCAT able to deliver the most deterministic response among all industrial real-time Ethernet systems dealt with in this literature review thus far. For a telegram of length 122 µs and at 44% bus load, results in [37] showed that EtherCAT had a cycle time of 276 µs. In comparison with the cycle times of other technologies, they revealed the following performances: Sercos III peaked out at 479 µs; PROFINET IRT at 763 µs; PROFINET RT at 6355 µs; and another popular protocol Powerlink V2 [45] at 2347 µs. In all this, EtherCAT still had 56% of its bandwidth to spare. It also has other advantages such as permitting incredibly precise synchronisation ($<< 1$ µs).

As mentioned, it processes frames on the fly like Sercos III, also using the summation frame principle, which is a major reason for their comparable fast speeds. It however utilises ASICs and FPGAs, and herein lies its chief disadvantage: the slave elements need to incorporate a specific hardware ASIC to implement EtherCAT [49], that is, extra hardware is needed for its implementation. Rinaldi [49] also noted how its data model is quite complex in nature.

2.1.5 OPC-UA

It is not exactly a protocol or technology on the same level as the others, but it can be used to work hand in hand with them, more specifically with Time Sensitive Networking, as will be explained.

Open Platform Communications Unified Architecture (OPC UA) is a protocol maintained by the OPC Foundation. As its name suggests, it is vendor-independent, unlike its predecessor, simply OPC, whose communication model used a Microsoft Windows proprietary technology: COM/DCOM. In fact, this is the reason for its very inception; to address the need to unify industrial communication systems. It will allow data exchange and interoperability through these systems, from sensor and actuator levels to central servers and clouds [50].

It transports data in two ways: firstly, through the client-server model, and secondly, through its own OPC UA PubSub (short for publish-subscribe). The client-server mechanism follows a request / response pattern to access information from the server, whereas the PubSub model is guided by a Publish / Subscribe pattern, where the server informs the clients rather than a client needing to pull data from the server. PubSub enables a many-to-many connection between the servers that publish information, and the receivers that subscribe to those that interest them. Should any changes happen to the publisher's data, the receivers that subscribed to it are automatically informed [51]. It is this PubSub that is developed and used alongside TSN to provide real time communications, to address the limitations of OPC UA with critical real-time processes [50].

The OPC Foundation is carving out a path to becoming "the industrial interoperability standard", as is stated in its logo. Over the years, it has signed numerous Memoranda of Understanding with many organisations and companies that produce and/or promote the use of industrial communication technologies and protocols, such as those mentioned in this review. These signed agreements pertain to the following matters: to intensify cooperation on Industrie 4.0; to ensure interoperability for IoT; to provide common interfaces for Industrie 4.0 and Internet of Things; and other issues as regards real-time data and open IT traffic with OPC UA [52]. Drahoš et al. [50] have shared that the initial results of these agreements was the distribution of the communication space in the following manner: these successful industrial Ethernet (IE) protocols hold the communication space at the field level, while the OPC UA protocol unifies vertical and horizontal higher production levels. With OPC UA, all these other communication technologies can be consolidated over a unified connection, as seen in Fig 5.

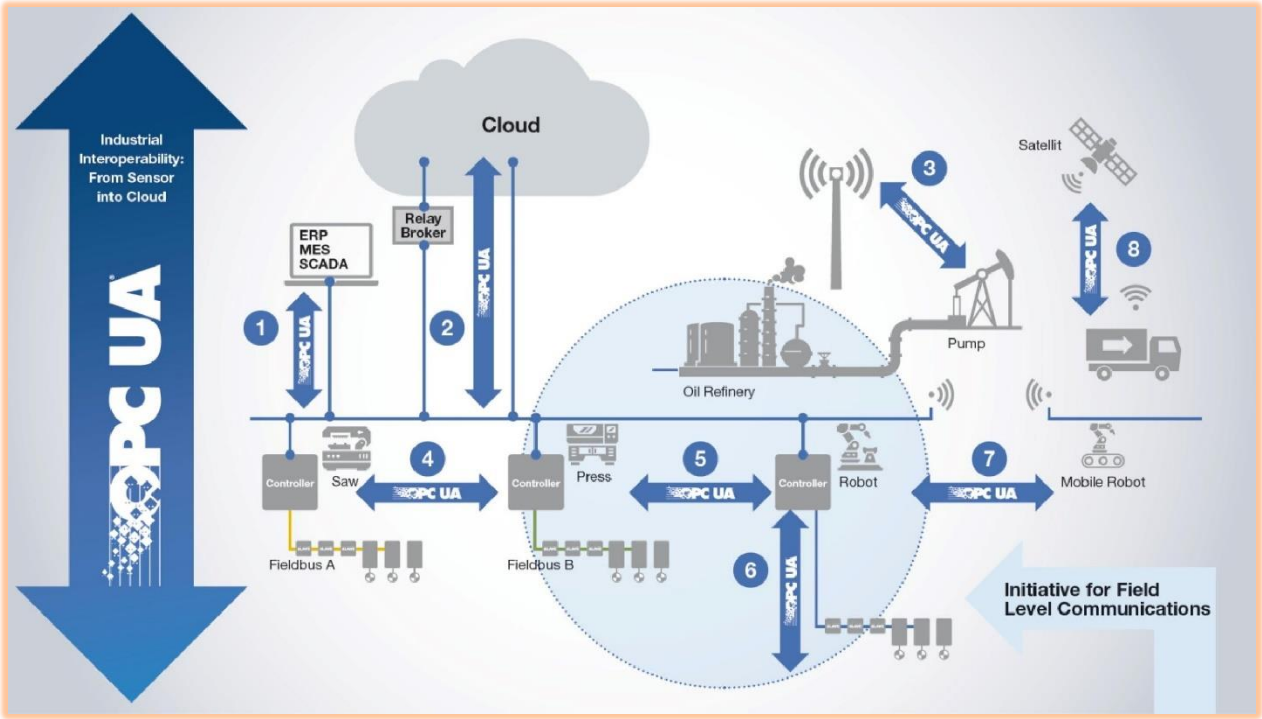

Fig 5: OPC Unified Architecture - from Sensor to Cloud [53]. *1 – IT / OT (Operational Technology) Communication, 2 – Cloud Integration, 3 – Secure Remote Access, 4 – Local OT Communication, 5 – Controller to Controller, 6 – Controller to Field Device, 7 – Wireless Integration (5G), 8 – Future Ready.*

Although it is stated that OPC UA agreements left other IE protocols with a kind of dominion over the communication space in the factory floor, more recently, the Foundation has sought to extend OPC UA's standards and specifications, including TSN, down to this field level. With the help of its new task force and various working groups, it hopes to provide vendor-agnostic end-to-end interoperability down to field level devices for all relevant industrial automation use cases [54]. Zimmerman et al. investigate this successful integration of IE technologies and protocols for the use case of skill-based field-level control in automation systems in their work in [51].

Because OPC UA is more than just a communication system (it is a standard for data exchange and semantic interoperability) [50], the writer believes that, employed together, OPC UA PubSub and TSN are two standards that will set the bar in the near future for first-choice industrial communication solutions.

Question 2: How do these technologies compare to each other? Can they co-exist with TSN?

2.2 Comparison of Industry Real-Time Technologies

To compare the response times of these fieldbuses more visibly, the writer replicates the table in [45, Fig. 1], shown as Table 1.

Table 1: Real-time comparison of the various real-time methods [45, Fig. 1]

Organisation	**Response Time (for 100 axles)**	**Jitter**	**Data Rate**
EtherNet/IP CIPSync ODVA	1ms	<1 ms	100 Mbit/s
Ethernet Powerlink EPSG	<1ms	<1 ms	100 Mbit/s
Profinet-IRT PNO	<1ms	<1 ms	100 Mbit/s
SERCOS-III IGS	<0.5ms	<0.1 ms	100 Mbit/s
EtherCAT ETG	0.1ms	<0.1 ms	100 Mbit/s

With aid from the experiment documented in the online Industrial Ethernet Book (IEB) [28] – the self-proclaimed "Directory and Information Source for Industrial Ethernet and Embedded Internet" – certain key categories were considered for performance measurement. The IEB table considers the case of an application and its real-time behaviour in a scenario where it is synchronously controlling 100 axles with these different protocols and technologies employed. It shows how they fared out. The two criteria whose performance measures are analysed are the response time (cycle time) and the jitter (or variation in response time).

It needs to be mentioned that end users are not ultimately concerned with the real time method with the highest performance output, but their choice is based rather on whether a method can support their application requirements [28]. So, a response time of a few milliseconds thought to be dire in certain applications might be quite tolerable for other scenarios such as the control of multi-axle drives in the case mentioned. Which leads to the next point: identifying different kinds of scenarios needing real time communications, and the exact specifications of their needs.

In their work towards time-sensitive applications support in 802.11ax and 802.11be EHT (Extremely High Throughput) [55], the 802.11 RTA (Real-Time Applications) Interest Group understood that although the average latency in 802.11 can be quite low, other crucial parameters may vary because of congestion; these being: worst-case latency, jitter and

reliability. They thus developed the following use cases, requirements and solution directions, to be considered for work on 802.11be in a bid to render it TSN capable, shown in Table 2.

Table 2: Application Requirements for 802.11be [55]

Use cases		Intra BSS latency (ms)	Jitter variance (ms)	Packet loss	Data rate (Mbps)
Real-time gaming		< 5	< 2	< 0.1 %	< 1
Cloud gaming		< 10	< 2	Near-lossless[5]	< 0.1 (Reverse link) > 5 Mbps (Forward link)
Real-time video		< 3 ~ 10	< 1 ~ 2.5	Near-lossless	100 ~ 28,000
Robotics and industrial automation	Equipment control	< 1 ~ 10	< 0.2 ~ 2	Near-lossless	< 1
	Human safety	< 1 ~ 10	< 0.2 ~ 2	Near-lossless	< 1
	Haptic technology	< 1 ~ 5	< 0.2 ~ 2	Lossless	< 1
	Drone control	< 100	< 10	Lossless	< 1 > 100 with video

In doing a flat comparison across the existing commercial technologies discussed so far, EtherCAT is seen to be the most capable technology for factory automation. It quite ably records minimal latency across the performance measures.

The main drawback of using these standard technologies stems from the fact that they are not interoperable; they are proprietary solutions (that is why the Internet Engineering Task Force (IETF) has a group dedicated to supporting deterministic networking, providing standards for the same, but over a single technology). They are not compatible with one another as is [23]. TSN can integrate with these open technologies and work alongside them, doing so through various interfaces such as gateways and couplers. Not only does TSN offer real-time capabilities that exceed current IE protocols [18], but since TSN is a standard, it can be applied by these different proprietary technology vendors so that the technologies from different vendors can work hand in hand. The previous section described how TSN is expected to interwork with OPC-UA specifically. Furthermore, per [6], [18], [23], TSN is an IEEE

[5] When examined alongside the requirements listed by the IEEE 802.11 Topic Interest Group noted by [33], "near-lossless" and "lossless" seemingly correspond to packet loss rates of < 10^{-7} and < 10^{-9} respectively.

communication standard that allows interoperability between standard-conformant industrial devices from any vendor. Vendors everywhere conduct interoperability testing to ensure an open platform for data exchange within control systems.

Question 3: What is the current state of TSN?

2.3 Review of Time Sensitive Networking (802.1Q)

2.3.1 Wired TSN

TSN technology was not conceived purely out of a need to achieve faster network speeds and lower latencies. It was understood that there is more that goes into the goal of reducing overall network delays. Ethernet as the precursor to TSN was quite capable of realising great speeds, and to this day it is able to keep up with the demands of most networks, reaching 100 Gbps speeds. What prompted and motivated the conceptualisation of TSN though is the fact that Ethernet lacks the determinism needed to guarantee Quality of Service, or QoS, on the end-to-end network flows [3]. On its own, Ethernet is not able to ensure a predictable end-to-end delay, or in other words, deterministic messaging, especially needed by critical applications. So then, as regards using stand-alone Ethernet for ULL (Ultra Low Latency) networks and applications, though it is a fast technology, it fails on certain accounts. As per Nasrallah et al. [3], these are:

1. a lack of QoS mechanisms to deliver packets in real-time for real-time applications;
2. no feature for a common time and synchronisation among network elements;
3. no mechanisms to manage network resources, like bandwidth reservation, for example;
4. no mechanisms to enforce policies, like packet filtering that ensures a guaranteed QoS level for the end-user.

Its standards have been discussed in earlier sections, but to briefly address the above aspects, they are here presented again.

TSN uses IEEE 802.1AS, a profile of IEEE 1588, to enforce a network-wide reference time to synchronise the clocks in its end-stations and switches [56]. This is its first defining attribute, and a staple for time-sensitive communications. IEEE 802.1Qbv affords TSN the ability to apply a global schedule through the network [5], where the scheduling mechanisms prioritise critical traffic flow over other lower priority traffic kinds. With these synchronisation and scheduling capabilities, TSN is able to provide deterministic latency [5].

Even though messages are set to transmit within fixed windows, there is the chance that some non-urgent messages may take longer to transmit. Through IEEE 802.1Qbu's frame pre-emption mechanism, critical messages can interrupt those non-urgent messages mid-flow

when they need to transmit, and after they are done, the previous flow will pick up where it left off. Finally, it has several protocols that allow it to control, reserve and allocate resources; as listed in [3], some of them are: IEEE 802.1Qca Path Control and Reservation (PCR), Resource Allocation Protocol (RAP), and the IEEE 802.1Qat Stream Reservation Protocol (SRP). This is how TSN establishes QoS for real-time applications.

Fig 6 shows the TSN standards summarised, as illustrated by Szancer et al. [41, Fig. 1], the main ones having been described earlier (IEEE Standard 802.1AS – timely synchronisation, IEEE Standard 802.1Qbv – scheduling mechanism, and IEEE Standard 802.1Qbu – frame pre-emption).

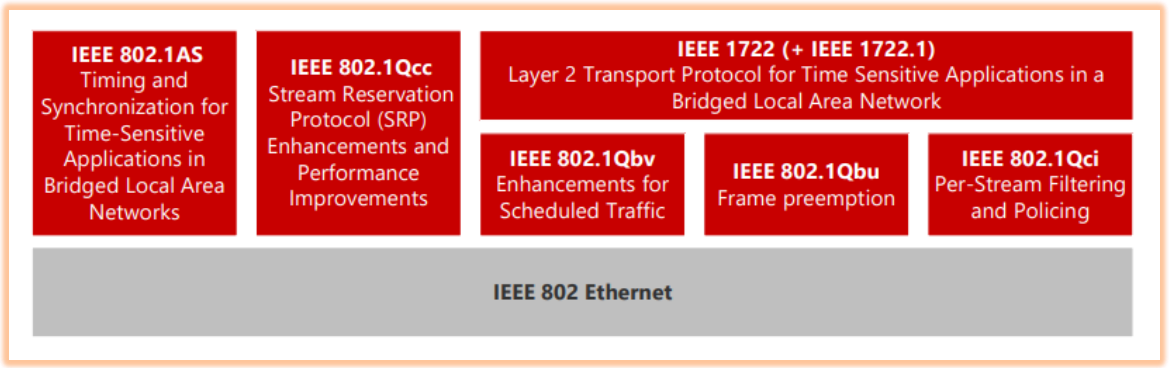

Fig 6: Time-Sensitive Networking standards [41]

Other standards which work towards reliability include **802.1CB** [19] – Frame Replication and Elimination for Reliability (FRER). As its name suggests, it introduces a redundancy aspect by dividing a stream into member streams with replicate packets, all headed for the same target destination. After which, it deletes the duplicate frames when the original is received successfully.

2.3.2 Wireless TSN

Some parties have thought of its (TSN) wireless actualisation, the most eminent being [24] and [25].

Bush and Mantelet [24] ideate a roadmap that explores how to leverage state-of-the-art TSN architecture into a wireless design. In their vision, they develop a theory of operation and identify technology gaps. They identify pivotal challenges that they foresee in each stage in their roadmap. For example, [24] acknowledges the fact that there are no known wireless implementations to manage the retransmitting of failed frames so that they fit inside the scheduled transmission window of their particular message queue. (This is not a problem for IEEE 802.1Qbv in wired TSN as scheduling mechanisms implemented in switches can strictly control the channel to ensure that frames get successfully transmitted in their time windows.) In their review, they provide a thorough description of how this transformation from wired to

wireless in industry can be broken down into five key phases. The suggestions they provide in overcoming some of the challenges that they highlight to this transformation may be considered vague by some. However, in the opinion of the writer, their main purpose was merely to shed light on a way forward.

The next sections show more work done for wireless TSN, but are more specific to the type of technology in use, namely: Wi-Fi and 5G.

2.3.3 Wi-Fi TSN

In a presentation Khorov [25] did for the Wireless Next Generation (WNG) Standing Committee Meeting, he introduced a novel approach for effecting Wi-Fi TSN. The major reason, as brought forward in [25], that accounts for Wi-Fi not being time sensitive is the packet delay. He attributes this delay to certain factors, namely queueing and packet transmission, and shows how they can be managed. His ideas are twofold: before all else, to stop the transmission activities of non-TSN frames when a TSN frame shows up; second of all, to reduce the channel access times and protect transmissions from collisions. A resulting dual-radio approach was presented to actualise these ideas. A primary radio would handle normal transmissions, while an auxiliary radio would inform neighbouring Wi-Fi stations to stop any ongoing non-TSN transmissions when this became necessary. These two radios would operate in different bands. After this proposal to the IEEE 802.11 Working Group, he later published his findings [33]. In [33], he and his peers spoke more extensively on the two ideas. It proposed traffic identification with the possibility to define new classes that the Wi-Fi stations could differentiate between; especially a class for real-time traffic. This would aid in his second idea of lowering the channel access time.

Khorov managed to draw attention to the problem of meeting the delay requirements of TSN over IEEE 802.11 Wi-Fi, as is reported in the meeting minutes that followed [57]. In preparing for TSN-capable 5G, dealing with these requirements to ensure the accepted 1 ms delay for low latency communications is an understandable concern. Some open issues were identified, chief of these likely being how to save partially transmitted data so that it can re-engage in transmission after the priority TSN frames are sent, as is the case with frame pre-emption in IEEE 802.1Qbu. Although the panel found interest in the matter raised on wireless TSN, they had reservations with the proposed solution, detailed here [57]. For one, it was identified that it cannot guarantee QoS in the unlicensed band. Manufacturers would have to support this solution by making Wi-Fi stations that would voluntarily give up the opportunity to transmit in the presence of a TSN transmission. As a result, some transmissions can be determined and managed, while others cannot. It became apparent that there remained underlying issues of latency in the approach. In their conclusion, they questioned whether it can be implemented. Despite its shortcomings, it served in painting a good picture of the barriers to overcome to support wireless TSN.

Seijo et al. [58] proposed a hybrid solution which combined wired TSN with their novel communication system that uses 802.11g Wi-Fi, to extend TSN features into Wi-Fi in an interference-free environment. Their solution was developed in a simulation environment using the OMNeT++ tool (a discrete event simulator to be introduced in Section 4). They have defined a new Physical (PHY) layer and a TDMA MAC layer. However, their solution does not support IT traffic, and needs further work to be able to do so. They suggest a way of doing this would be to make it backward-compatible with standard 802.11 Wi-Fi which it currently does not support. Instead, it supports periodic URLLC (or PURLLC). However, since their system and Wi-Fi are not completely integrated, their model has certain downsides, such as the lowered amount of traffic that can be transmitted by Wi-Fi nodes compared to normal use implementations of Wi-Fi. This entails prescribing a lower MTU (maximum transmission unit) so that only small frames can be allowed to transmit, and also having to manually configure each access point for these design characteristics. Another major downside is in the distinguishing feature that allows their model to be compatible with 802.11 nodes: these Wi-Fi nodes must not be allowed access to the channel during transmission periods reserved for real-time traffic. This is very similar to what Khorov proposed in [25], and later in [33], discussed in the previous section. To overcome these, they propose amendments to the 802.11 standards to accommodate certain characteristics of their system.

Continuing from this, Genc and Del Carpio [59], on recognising the amendments needed, actually modify the MAC layer in their solution to cater for the QoS enhancements needed by Wi-Fi for TSN, for the downlink in industrial automation. This modification includes defining a new access category in the Wi-Fi MAC layer to handle higher priority traffic, also an idea that Khorov had in [33]. They proved, by means of simulation, that introducing a new QoS access category would significantly improve the delay and jitter performance. However, they could not strictly guarantee the reliability as required by TSN applications: although the majority of packets performed well, the rest were not delivered timely, and in order to prevent the delay and jitter in the network from being increased, they had to be dropped. This contributes to higher packet losses, which goes against the goals of reliability. They conclusively state that their model is suitable for those use cases in industry which do not need very high reliability with reference to latency, delay and packet loss.

Wireless TSN is so appealing because of the added flexibility in the non-wired domain that it can provide, especially in IIoT devices: extended roaming capabilities for automated guided vehicles (AGVs); increased mobility for industrial robots; not to mention the reduced costs for cabled infrastructure. Another major reason is the ultra-reliability and low latency communications (uRLLC) that comes with applying it with wireless technologies like 5G. Further work has gone into speculating on the future of TSN, some more notably going on to describe a possible 5G-TSN architecture. The works surrounding such an architecture will be reviewed hereafter.

2.3.4 5G-TSN

The 3GPP (3rd Generation Partnership Project), in its integrated system architecture, has two approaches on how 5G TSN can be achieved, of which the most favourable involves extending the 5G system (5GS) so that it can connect to the IEEE TSN network as just another among its IEEE TSN-capable bridges [60]. This representation would render the entire 5GS, from end to end, to be seen as a single logical (or virtual) IEEE TSN 802.1AS time-aware bridge, as Fig 7 shows. To support TSN in this way, 5G would need modifications to its radio access network and core elements. The specific areas that 5G is currently still limited in, as presented in [60], include: firstly, some features to provide for the enhanced scheduling in TSN that has transmission windows as small as 10-20 µs accessed in the time domain (while in the 5GS RAN (Radio Access Network), this currently sits at 100 µs or larger, accessed in its frequency domain); second, a set of features to ensure the radio performance in terms of latency, reliability and capacity. According to Mannweiler et al. [60], enhancements are needed to guarantee the availability of radio resources in 5G cells that would fulfil these latency and jitter requirements. Jitter in this regard can be understood to influence predictability (it has been expressed in earlier sections how there is a direct relation between predictability and reliability, and the overall efficiency of a network – which is the end-goal of time-sensitive communications). Additionally, time sensitive traffic classes need to be accommodated in 5G QoS profiles, as outlined in the meeting in [61]. Some of the enhancements for 5G TSN integration is included in 5G's release 16, which was scheduled for completion in the 3rd quarter of 2020, with further enhancements expected in Release-17, sometime in 2022 (delays were experienced because of the pandemic). This includes time synchronisation for the UEs (user equipment) [62].

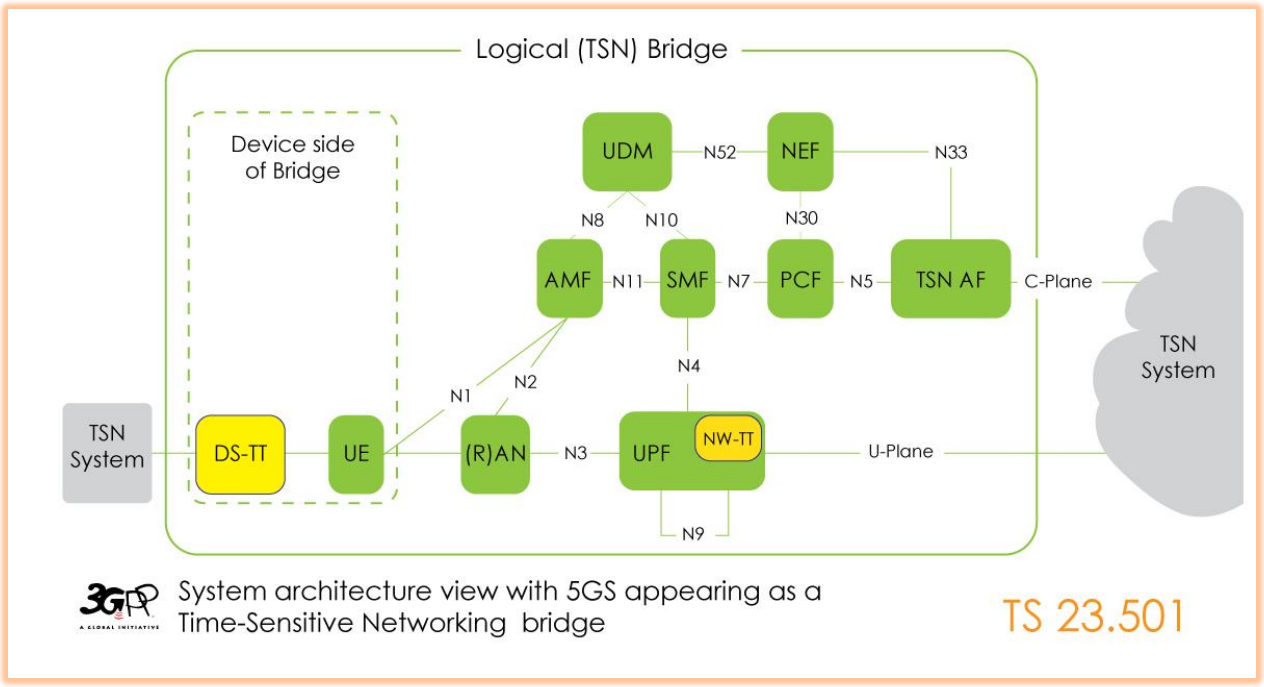

Fig 7: 3GPP 5G & IEEE TSN Integration [62, Fig. 1]. *5GS – 5G System, AF – Application Function, AMF – Access and Mobility Management Function, DS-TT – Device-Side TSN Translator, NEF – Network Exposure Function, NW-TT – Network-Side TSN Translator, RAN – Radio Access Network, SMF – Session Management Function, UDM – Unified Data Management, UE – User Equipment, UPF – User Plane Function.*

Neumann et al. [63] present different scenarios to combine IE with 5G mobile networks, and included is this same discussed concept of the transparent integration of 5G and TSN. Their scenarios are limited to discussions, and not actual implementations. They conclude that the idea of transparent integration may be beneficial, although its performance and usability still need to be investigated.

Such an investigation is presented in the analysis in [64] on 5G-TSN integration, more specifically focusing on quantifying the 5GS bridge delay for the case of a closed loop control industrial application. It aimed to discuss the open technical and research challenges to support low latency and determinism in 5G-TSN; it did not end with an actual model. Rather, it presented 5G-TSN integration measures that can be used to identify further limitations to be looked into in future work.

Martenvormfelde et al. [65] have also recognised how the industry has to make a trade-off in having wired systems with latency and reliability measures that suit their real-time applications, at the expense of the degree of mobility offered by wireless systems. They attempt to address how this can be overcome by including 5G into TSN as a transparent bridge, as discussed previously, using a closed loop control simulation based on OMNeT++ and NeSTiNg, two important simulation modelling tools which this very work is also based on and which shall be presented in Section 4. They maintain that with 5G, many issues of

interference will be avoided since it functions in licensed frequencies. They also believe that, in time critical applications, merging 5G with a wired real-time communication technology, like TSN is a key to realising the true vision of Industry 4.0. Their model mainly aims to investigate the challenges of 5G-TSN integration because they argue that 5G development is currently at a state where the availability of devices supporting Release 16 is not granted. In [65], simulative approaches are most ideal for large scale networks due to the reduced time for deployment and their flexible nature. Their model utilises a lightweight 5G user plane, although it is still under development. It still needs to be extended in relation to 3GPP specifications, for example the inclusion of IEEE 802.1Qbv for priority scheduling behaviour [65]. Once 5G-TSN standardisation is at a later stage and further developed, more complete solutions may follow.

Ray et al. [66] define a setup which includes an Ethernet switch implemented by a PC motherboard with FPGA and DSP (Digital Signal Processor), connected at the end point to a 28 GHz 5G NR link for transmission and reception by another 5G NR radio link, in efforts to realise 5G TSN synchronisation in the context of non-public networks. The use case addressed is in smart systems such as smart homes and vehicles. They report having achieved successful 5G-TSN synchronisation, although a review of their system reveals that it is rather different. They instead have achieved wireless synchronisation of videos between two remote locations, over wireless links, which they maintain, can aid in streaming live TV and other IoT applications in smart homes. In the next setup they want to implement, which is a work still in progress at the time of this writing, they propose to replace the Ethernet switch with a TSN system.

Gundall et al. [67] present another work that also deals with synchronisation. They introduce a concept for integrating the time synchronisation standard of TSN, that is, IEEE 802.1AS, with 5G to meet the needs of mobile use cases in industry. In their testbed, they have developed an alternative solution to that of 3GPP's (previously detailed) to achieving this, since there is currently no complete 3GPP Release 16 hardware available to implement their prescribed solution. They report how theirs is also different in that it remains compatible with 4G. Their setup consists of a full 5GS, many UEs, and a Reference System with an 802.1AS Grandmaster device, all hosted in a testbed that uses 3GPP 4G as an example. The 5GS has its own synchronisation mechanism from the TSN system, but it merely uses the offset of TSN from the Reference System to synchronise its devices. Each gNB (or "next generation" NodeB, which is how to refer to the 5G base station) in the 5GS synchronises the UEs connected to it, using 5G's own synchronisation signals: primary synchronisation signal (PSS) and secondary synchronisation signal (SSS). OPC UA is used to distribute these signals [67]. This alludes to the fact that many UE's can subscribe to the same server, as per Section 2.1.5.

In concluding, 5G can complement TSN and is a good base to begin with to drive efforts into producing a wireless TSN solution.

5G standardisation for the aforementioned enhancements and others is ongoing. At the time of this writing, there hasn't been a working implementation of a wireless TSN-compliant service that has been standardly deployed for the industry or any other use case. All working implementations of TSN are of the wired variety.

2.3.5 DetNet – An Alternative Deterministic Low-Latency Networking Standard

As mentioned in Section 1, there are other networking standards aside from TSN that can be used to architect deterministic ULL networks. Prevalently discussed among these in academic circles is the IETF's DetNet, or Deterministic Networking. To achieve this determinism, it is designed to operate with high reliability and millisecond latencies [26], and in this regard, it is similar to TSN.

It differs, however, in that, where TSN ends off in the OSI (Open Systems Interconnection) model's Layer 2, DetNet extends its support to cover IP routing in Layer 3 [3], [24], [26], and, according to Yang et al. [26], if needed, it will extend into the higher layers of the OSI model. Because it operates in these layers, it also has to pay more attention to security to defend against cyber-attacks, unlike in TSN.

As per Yang et al., other features of DetNet include the keeping of one common time, and zero congestion loss which adds to reliability. Like TSN, it synchronises its network elements using the IEEE 1588 mechanism and IEEE 802.1AS (which is based on IEEE 1588v2 PTP). And to deal with congestion, it uses queuing algorithms and packet pre-emption, much like TSN too. DetNet goes a step further though in that it not only guarantees an upper bound on latency, but it employs methods to fix a lower bound as well, the purpose of which is to minimise jitter [26].

Another fundamental difference identified in [26] is in the progress made in both research directions. Although great strides have been made by the DetNet Task Group (TG) into developing DetNet's standards, it is as yet largely unfinished. Compared to the TSN TG, whose series of networking standards have been published for years now, DetNet documents presently remain as internet drafts, like those in [68]. They still require further work.

As per Nasrallah et al. [3], the goals of TSN and DetNet are aligned similarly; specifically, they aim for deterministic worst-case bounds on latency, on jitter or packet delay variation, and also for extremely low/zero packet loss. Many other ULL technologies and standards exist, albeit they have different goals other than deterministic messaging like TSN and DetNet.

This work does not deal with DetNet. It is presented here merely to highlight alternatives to TSN.

Question 4: How is TSN currently employed by different leaders in industry innovation?

Different companies have come up with varying methods to implement TSN with their own TSN solutions, most notably in [17], [18], [69], which will now be examined.

2.4 TSN in Industry

Intel and **TTTech** choose to implement TSN over System on a Chip (SoC) FPGAs for use in industrial systems chiefly because of their high speed (important for minimizing time delays) and inherent reprogrammable nature; they can be reprogrammed to implement new standards in any configuration [18], a feature which they hold over other silicon-based application-specific integrated circuits that industrial equipment manufacturers may use. Other benefits include workload consolidation and acceleration, as well as I/O flexibility since FPGAs can be configured for TSN and still allow for other Industrial Ethernet (IE) protocols to be implemented on the same device [18].

As long as the key standards are adhered to, TSN in its implementation proves to be quite flexible. Another industry giant, **Cisco**, chooses to focus on said key features in its own TSN-capable devices.

In their white paper, the authors in [69] discuss how, in order to address the requirement of the networking devices in a TSN communication having a shared concept of time, the Precision Time Protocol (PTP) is used. It comes in two profiles: **IEEE 802.1AS** and **IEEE 802.1ASRev**. Furthermore, their solution architecture comprises five chief components:

i. **TSN flow**, which is a single communication between networked devices that is characterised by a unique ID and strict time demands;
ii. **End devices**, the talkers and listeners which respectively initiate a TSN flow and receive the message contained herein;
iii. **Bridges** or **Ethernet switches**, which operate differently from normal bridges in that they transmit and receive Ethernet frames of a TSN flow according to a schedule;
iv. **Central Network Controller (CNC)**; and lastly,
v. **Centralised User Configuration (CUC)**.

They have adopted a centralised approach to network management, as documented in the IEEE standard **802.1Qcc** and illustrated in Fig 8 [4, Fig. 1].

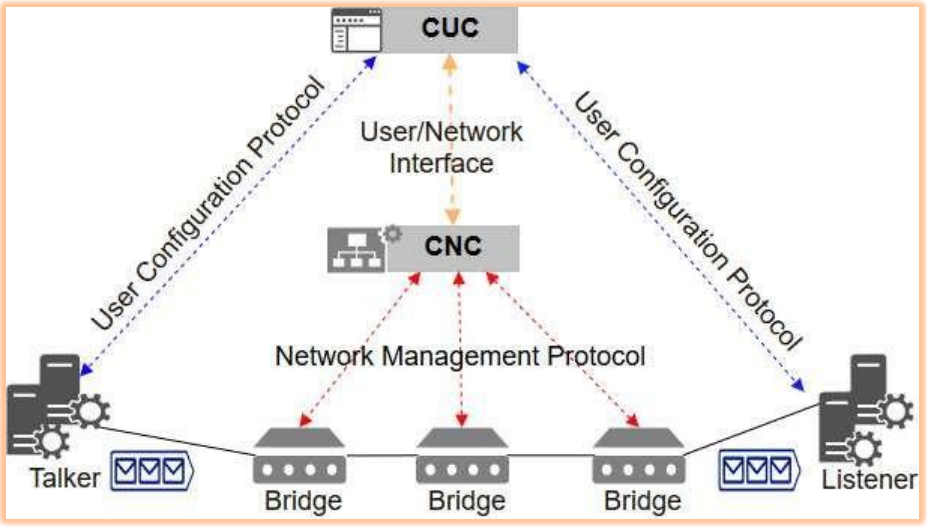

Fig 8: IEEE 802.1Qcc Central Configuration Method [4, Fig. 1]

According to the solution described in [69], the CNC is an application that resembles a proxy in its mode of operation; as also evidenced in [56], it constructs the schedule which bridges rely on for transmitting frames. Moreover, it assigns streams to egress port queues, and schedules the transmission of each of the frames on these streams [56]. Different vendors of TSN bridges have developed different CNC applications to control their bridges, so it is only natural that Cisco has its own. As for the CUC, it is an application that acts as an interface between the CNC and the end devices [69]. Different TSN flows will generally have different requirements for communication, and the CUC intercedes on behalf of the end devices with the CNC so that it meets these requirements. Like the CNC, it is also vendor-specific, and it is on the part of the vendor of the end devices to develop a CUC to represent them.

To illustrate how scheduling, regulated by **802.1Qbv**, is paramount in maintaining TSN flows, Cisco used an analogy of trains operating on schedules [69]. But how does scheduling actually work? Just as trains move from station to station, one can imagine TSN frames moving from a bridge to a bridge over a link between them. Non-TSN frames are accommodated as well, but they can only use the link at times when there is no TSN communication scheduled to use the link. As for moments when an urgent unscheduled TSN message is needed to be relayed but there are non-TSN frames currently using the link, the **802.1Qbu** standard is used for pre-emption of frames. All of these flows are synchronised to a common time by the standard **802.1AS**, or in future, **802.1ASRev**.

As mentioned, each flow will have a unique identifier. For the Cisco implementation, this consists of: destination MAC (Media Access Control) address, VLAN, and Class of Service[6]

[6] Class of Service – expressed as a 3-bit field in the Ethernet frame header that allows to group the various kinds of traffic into different classes each with its own level of service priority. It provides traffic management like QoS, but unlike QoS it does not guarantee the level of service (that is, bandwidth, latency). It offers only a "best effort".

(CoS). In other sources like [3], each flow is generally defined by the Priority Code Point (PCP) field and the VLAN ID (VID) in the 802.1Q VLAN tag in the Ethernet header.

The prime benefit of implementing all these components is that it would enable customers to integrate and run multiple services over a single physical connection, while guaranteeing QoS.

These companies have one thing in common; they rely on a physical Ethernet infrastructure to service their industrial communication needs. Not much work has been done as far as realising precise-time machine-to-machine communications over wireless means.

2.5 Chapter Summary

The chapter just concluded has presented a state of the art regarding achieving deterministic communications for the industrial use case. This included the technologies in wireline environments, of which the bulk of literature leans towards wired solutions as the industry is mostly occupied by field-bus technologies. And further, the review touched on the current work being theorised and investigated for the wireless case. TSN was presented as a viable solution to support time-sensitive traffic going forward. It was contrasted against other methods in its design and mode of operation. Many of the current solutions for determinism on the factory floor, including TSN, have been implemented over standard Ethernet. The case of deterministic wireless infrastructure has been identified as an open research problem, and this thesis shall place emphasis on this gap.

This chapter, in its content, additionally helped to meet the first objective of the work, which also in part answers the first research question: how wired and wireless channels differ as regards industrial communications. It reviewed the differences in the mediums for communications, along with the different approaches and technologies used and how they make use of those mediums. The next chapter goes on to show in detail how this wireless TSN is envisaged to look like, in terms of its requirements and the design considerations.

3. Requirements and Design of Wireless TSN Simulation

In the previous chapter, works related to achieving deterministic communications for the industrial use case were discussed. This chapter discusses what goes into visualising a wireless TSN environment. It provides an in-depth look into TSN over wireless: beginning with the choice of wireless technology, then follows the hindering factors, it further elicits requirements for a wireless implementation, next, the design considerations brought to perspective, and ends with the envisioned mode of operation of the proposed wireless design.

3.1 Choosing the Wireless Technology for the TSN Model

The previous section's review discussed the few wireless technologies that TSN has been proposed to work alongside. 5G was discussed as a possible option because of the many possible use cases it has potential for in future. It is indeed an attractive technology in terms of its timeliness and especially in its URLLC aspect, which drives 5G's development. But ultimately, this does not equate to effective time-sensitive communications. More to the point, 5G is not particularly ahead of other wireless technologies because it also lacks the essential features required to handle synchronisation and scheduling. Several enhancements are required for it to be able to accommodate TSN, including time-sensitive traffic classes to accommodate 5G QoS profiles, among others [60], [61]. Even according to the 3GPP definition, this has to be timed and rolled out in stages with future releases [62]. In this regard, 5G may not be a suitable option for the proposed model. As for 4G, the facets of its technical profile are also a source of its falling short of realising time-sensitive communications.

At the time of development, the few works written up on wireless time-sensitive communications have shown considerable progress using Wi-Fi, like in [59], while those using 5G are developed in relation to 3GPP specifications, which is still ongoing as evidenced in [65]. It stands to reason that more tools were found that would readily support experiments in this wireless technology. Therefore, Wi-Fi is the preferred choice for the proposed TSN model.

Presently there aren't any working *wireless* implementations such as the one proposed in this work. To reiterate, there are, deployed across the industry in the world, TSN-compliant services, but there has not been any standardly deployed or recorded implementation of TSN for the industry or any other use case, that is based on a wireless medium as part of the architecture. In effect, all working implementations of TSN are of the wired variety.

3.2 Factors Affecting Determinism in a Wireless Network Infrastructure

Why is Wi-Fi generally unsuited for such communications? There are certain factors that make Wi-Fi not time-sensitive. Identifying these factors provides a starting point in answering the second research question – what are the factors that affect the reliable transmission in wireless TSN?

The Wi-Fi Problem

According to Khorov [25], the matter in question has to do with the end-to-end delay of packets, something that can be attributed to two main sources:

$$Packet\ Delay\ =\ Queueing\ Delay\ +\ Packet\ Transmission\ Time$$

Eq 1

In relation to TSN, the queueing delay is a matter of how long the TSN packet should wait in a queue while those ahead of it are being transmitted and their successful receipt confirmed. As for the transmission time for packets, this is summed up by the PPDU (PLCP (Physical Layer Convergence Protocol) Protocol Data Unit) duration, or packet duration, and the channel access time. The channel access time could be especially problematic taking into consideration the fact that Wi-Fi makes use of unlicensed spectrum. A brief look into the most popular frequency range used that resides in this spectrum will help explain where the problem lies exactly, that range being the ISM[7] 2.4 GHz band.

Insomuch as spectrum is a limited resource, government organisations in nations across the world are responsible for its leasing and allocation to users. Among these users, those who are granted a license to access a portion of the spectrum are each given a designated frequency range and legal protection against interferers. Most users prefer to deploy their proprietary or other solutions in unlicensed spectrum, however, to avoid any license fee. They thus tap into the free 2.4 GHz frequency band of Wi-Fi, which means many devices share the same frequencies, and no protection against interference can be guaranteed. Since any device is able to communicate over this frequency range, Khorov [25] inferred that the channel could easily be occupied by other devices' transmission activities.

Furthermore, Wi-Fi is not even the sole technology that participates in spectrum-sharing: both Bluetooth and Zigbee technologies are preferred for their low hardware costs and low energy consumptions at the end-devices [70], and they both operate within the 2.4 GHz band. More and more devices each day incorporate these IoT technologies, which only adds to the congestion and interference in the air waves. There are other IoT technologies, apart from these, which leverage unlicensed spectrum, although they often operate at different ISM frequencies; two newer ones being LoRaWAN and SigFox. Cellular unlicensed spectrum

[7] ISM – the Industrial Scientific Medical radio bands are globally license-exempt and free to access, subject to certain constraints such as restrictions on the transmit power. Most technologies such as Wi-Fi and Bluetooth that operate in this band suffer from severe interference issues because of the sheer number of devices and signals that operate in these ranges of frequencies.

technology[8] is another technology that shares spectrum with Wi-Fi, although it claims to practice fair coexistence with Wi-Fi and other existing networks [71].

As regards TSN, implementing it over Wi-Fi is difficult to achieve because of the above-discussed highly indeterministic ISM channel. While it is perfectly fitting for other applications, it would normally be unreliable in an industrial scenario with inflexible timing demands to the order of milliseconds.

3.3 Key Requirements for Wi-Fi TSN

3.3.1 Requirements of the Medium

The use of the wireless channel for real-time applications deems that it has certain requirements, as identified from earlier sections.

In their study on enabling real-time applications over Wi-Fi networks, [33] were able to establish key requirements for a variety of use cases ranging from applications for entertainment to critical control systems. Their data is replicated in the table that follows. In many ways it resembles those identified by the 802.11 RTA Interest Group in Table 2 of Section 2.

Table 3: Requirements for some real-time applications considered by IEEE 802.11 Topic Interest Group [33]

Application	Latency, ms	Bandwidth	Packet size, bytes	Robustness
Emergency control	1 … 3	10 … 100 kbps	<100	PLR $<10^{-9} \ldots 10^{-5}$
Real-time video	3 … 10	0.1 … 20 Gbps	>4000	PLR $<10^{-7}$
Online gaming	10	0.1 … 1.0 Mbps	100 … 200	<3 consecutive loss, PER <0.1%
Regulatory control/camera input	10	10 … 100 kbps	<100	PLR $<10^{-9} \ldots 10^{-5}$
Supervisory	10 … 100	1 … 10 kbps	<100	PLR $<10^{-9} \ldots 10^{-5}$

PLR: packet loss rate; PER: packet error rate.

From the table, the following requirements can be elicited:

- **Requirement 1** – a reliable wireless channel. This translates to a high success rate for packet delivery (near lossless).
  A wireless option like Wi-Fi operates in unlicensed spectrum, as discussed. This means it oftentimes makes opportunistic use of the spectrum because of the multiple other devices operating in this band which access the channel randomly. More packets are dropped, and it becomes difficult to guarantee reliability.
- **Requirement 2** – low latency. Real-time applications require a delay of under 10 ms for most time-sensitive industrial communications. In other cases, relatively longer delays are tolerated.

[8] Cellular unlicensed spectrum technology is a term that comprises the cellular technology that exploits the unlicensed band spectrum. An example is the Long-Term Evolution (LTE) License-Assisted Access (LAA) technology, designed by the 3GPP [71]. Another is LTE-U (LTE-Unlicensed).

Longer latency times often come about as a result of packets being dropped needing to be retransmitted. An additional source of delay is in waiting for the channel to become idle before a Wi-Fi station can occupy it with its own transmission. According to [33], some transmissions can be rather long, with some reported as having an upper bound of 5 ms; legacy Wi-Fi stations cannot interfere with an ongoing transmission that they can hear, once it's begun.

- **Requirement 3** covers capacity. A steady bandwidth of capacity 100 Mbps, comparable with the data-rates that can be averaged by most fieldbuses (or within 100 Kbps as per Table 3), is required.
  Industrial applications do not require much bandwidth in their communications and non-wireline approaches such as Wi-Fi are capable of meeting the stated average since its present standard, 802.11ac (or Wi-Fi 5), caps out at several Gbps theoretically (actual speed reported being well under 1 Gbps).

3.3.2 Requirements of the Equipment and Device Components

At present, TSN-enabled devices can participate in all the activities of a TSN communication, such as electing a Grandmaster device if they can connect to the network through standard Ethernet.

For the proposed system though, and as is standard of a Wi-Fi communication, all devices will require, at minimum, one wireless radio or a WLAN card and an accompanying wireless interface to be able to transmit and receive Wi-Fi signals.

3.4 Key Design Considerations

After identifying the factors that affect Wi-Fi transmissions in time-sensitive communications in general, even beyond the scope of TSN, it is possible to begin considerations on the design of the model earlier proposed.

On the conceiving of said model, some concerns were brought forth based on the review of the literature in Section 2.3.2. The findings on wireless TSN, particularly those in [33], [34], [60], [61], point towards the need to include several enhancements in the present wireless systems in order to effect a time-sensitive communication that reflects the TSN standards. These modifications advocate for defining new standards and modes of operation, some of which will require nothing short of redesigning the physical layout of wireless devices to produce a new hardware solution as in [25]. All of these go towards realising the most complete solution of wireless TSN.

Since this would entail firstly a thorough consideration and definition of all necessary design aspects by the standards bodies and organisations before producing the new components, this work would need to follow a different methodology. Instead it was decided to leverage the present hardware components as they are and integrate these with the present wireless

capabilities as suggested by [24]. It may involve reconfiguring the components' layout or software to an extent so that tools that didn't need to talk to each other before can be used in a different manner from their prescribed modus operandi. This is so far the most feasible approach research-wise. Inspired by the roadmap envisaged by Bush and Mantelet [24], the writer conceptualises a different phased approach towards going about this method, with more details regarding each point following this high level breakdown:

1) Introduce wireless hardware
2) Configure the switch and access point interfaces
3) Design the interconnection between wired TSN part and wireless part
4) Establish a time-sensitive communication

Stage One: Introduce Wireless Hardware

The first stage makes use of wireless devices and components to replace their wired counterparts. The network connections made here reside in the physical layer of the OSI model, which is responsible for the actual physical connection between devices. This phase is the more straightforward since it consists mostly of replacing host devices such as PCs and workstations with devices that incorporate wireless LAN cards and a working Wi-Fi connection to transmit data as opposed to an Ethernet port and line. In this way, it will include specifying the radio frequency link, as opposed to the cable type. Aside from being responsible for all the physical interconnections between hosts and pc's, it also performs bit streaming. Because of this, it controls the bit rate, or the transmission rate, and defines other options such as the transmission mode. On this note, Wi-Fi transmits in unlicensed frequencies, which a lot of other devices are also using. That being the case, it opportunistically accesses the spectrum when it is free, and so devices only send data one way; they can either transmit or receive at a time.

As for the physical topology, further thought can be given to it after eliciting the requirements from the other stages.

Stage Two: Configure Interfaces

With the introduction of wireless hosts follows that of wireless access points that are used to carry the wireless transmissions and propagate these to other devices. In the OSI representation, inter-node transfer sits at the data link layer, while routing happens in the network layer. Since access points work with switches to forward frames, they operate in the second and third layers of the OSI model, that is, the data link and network layers. The data link layer performs "switching", which transfers a frame from one interface to another, within the same device, thereafter it can be forwarded out to another node. It is further subdivided into the Logical Link Control (LLC) and the Media Access Control (MAC). The upper and lower wireless MAC layers shall be configured so that they process the TSN frames in the same

manner as the Ethernet MAC, as they interface between the LLC and physical layers, as in Fig 9.

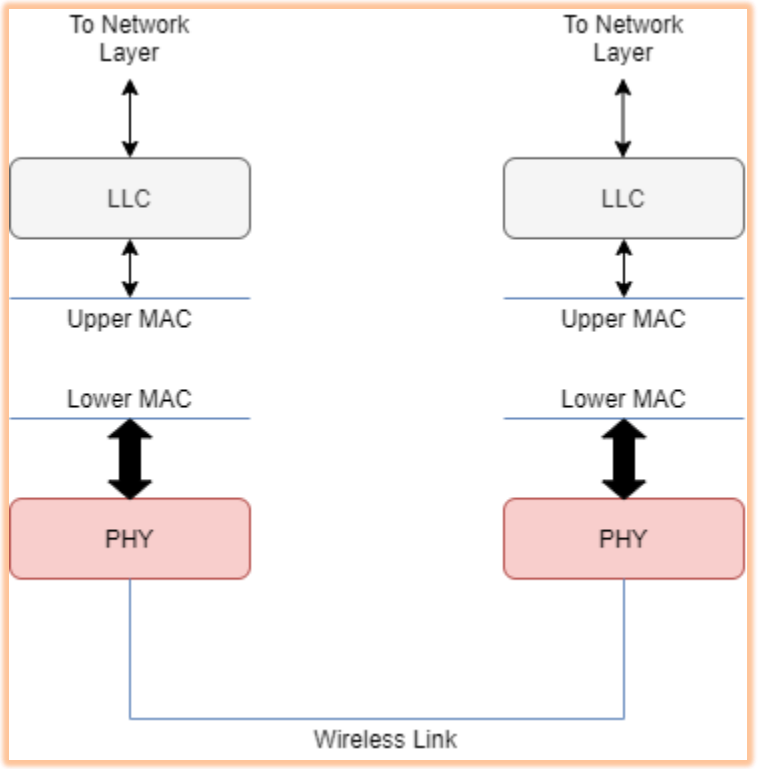

Fig 9: Upper and Lower MAC Layers

Because configuring interfaces means working in these two layers, the implementation will involve configuring the following parameters: MAC addresses, an address resolution protocol, a routing file containing forwarding paths and destination addresses, definition of gates or interfaces, and interface layers for the devices.

Stage Three: Design Architecture of the Interconnections between Wired and Wireless

This stage considers part of the third research question – how can one contend with the problem of transmitting TSN frames and non-TSN frames concurrently in the same network?

Essentially, it details the architecture that realises how the two mediums will connect with each other.

A program is needed for the control and management of frames, and ultimately, synchronisation. In wired approaches, this is perceived as the combination of two entities: the CUC and the CNC. In using a similar system for the wireless implementation, it is rational that it be connected to the switches and bridges instead of only to the access points because the current wired model has the authority to start and stop transmissions on schedule, whereas Wi-Fi has to wait for that opportunity since it leverages free spectrum. The TSN traffic will be

sent through a queue that will be configured as high priority, while the non-TSN traffic for other services is sent in queues set as low priority, according to PCP mapping conventions. They will both have to pass through Ethernet and then converted to be suitable for transmission through wireless radios, as in Fig 10.

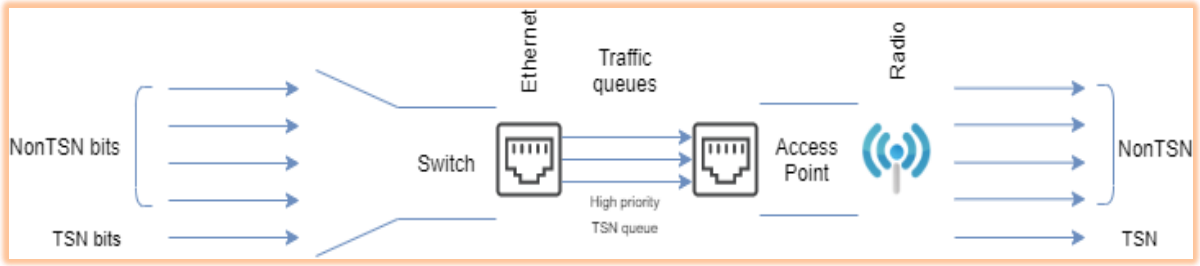

Fig 10: Wired Talks to Wireless

An alternative approach would be otherwise to dictate an entirely different mode of operation for the standard wireless controller or access point, which is currently beyond the scope of this study, as explained in the beginning of this Section 3.4. On arriving at this conclusion, it is necessary to reiterate Stage One, and understand that not all wired components are replaced, some are still necessary. And so, the physical layer now consists of these parts coalesced into the design: cabling, wired hosts and any peripherals. The functions of the CUC and CNC will be integrated into the switches, instead of in wholly separate applications.

These design points considered, the most logical approach points towards a hybrid set-up of both wired and wireless components working together, such as is rendered in Fig 11. Further, because these considerations suggest some specific degree of customisation, an open-source simulation program is a good environment to build the model in. A simulator would offer the opportunity to trial-run experiments of different setups, upon each modification of the code.

The physical topology is now considered; the implementation is modelled on the distributed topology, and not the centralised Qcc configuration from Fig 8. It has been altered this way to favour the installation of the mentioned wireless components. As per Bello and Steiner [6], the CNC and CUC may not be present as part of the network in operation at all.

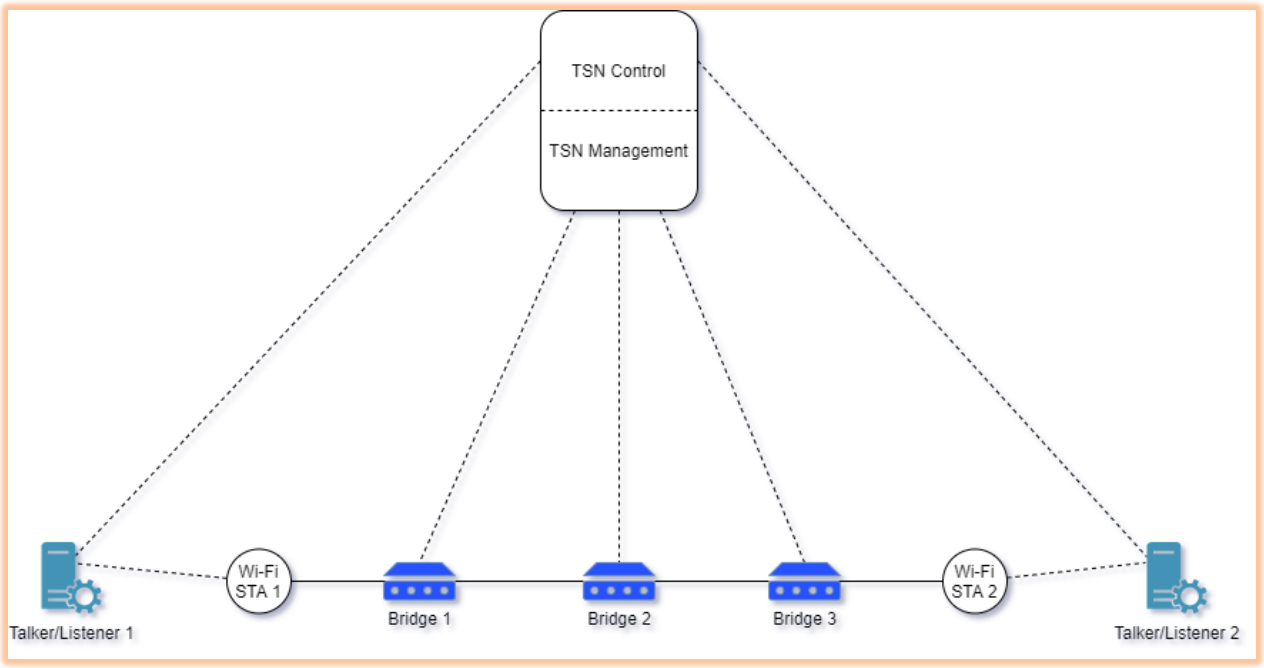

Fig 11: TSN Wireless Implementation. Dashed lines – wireless links, Solid lines – wired links, Bridges – TSN enabled, Wi-Fi STA – access points, TSN Control and Management – composed of CNC plus CUC.

Stage Four: Establish a Time-Sensitive Communication

After verifying that the individual components of the system have been configured, they now have to be made "aware" of each other, that is, they should be able to reach one other. Further, they should be able to establish communications and co-exist with these different communication classes, best-effort and critical, in a time-sensitive ecosystem. One in which the communication obeys rules for precisely timed delivery, and scheduled, synchronised operations.

As such, it is concerned more than anything with the transmission and reception of data. This involves working in the second layer (already covered), but more so the fourth OSI layer – the transport layer. This is because the transport layer coordinates the successful end to end delivery of information, which is what is meant by established communication. This layer comprises everything to do with the delivery of information and how much of it is sent across the network. The transport layer further involves the acknowledgement upon successful delivery of the message, and the re-transmission upon failed delivery or on receiving an erroneous message. TCP and UDP port numbers work here too. When a host device transmits a message to another host, this layer also dictates how long the host should have to wait for acknowledgement of successful delivery. When the segments that are transmitted are too large or too small, the transport layer performs windowing, another of its main functions, to contend with any problems that may arise because of this.

With this in mind, for the implementation of the TSN environment, the following settings need to be defined: schedule file or forwarding times to accommodate TSN, the transmission rate or data rate, wireless radio type, wireless LAN SSIDs (Service Set IDentifier) or channel identifiers, carrier frequency(ies), port ID or port numbers, MAC retries limit, different queues and queue priorities (to separate and prioritise the different traffic, that is, TSN and other), queue capacity, gates and gate controllers for the different queues to work based on the schedule, packet sizes, contention window parameters. Other parameters for the wireless device radios are important: transmitter power, receiver sensitivity, receiver SNIR (signal-to-noise-plus-interference ratio), values for interference and the communication range, among others.

Since this work is concerned with coordinating and modelling the flow of data, and not necessarily how it will be processed or used by the end devices, the first four layers of the OSI model are the main layers brought into use.

3.5 Envisioned Mode of Operation of the Wireless TSN Model

Projections can be made towards how the model will operate. It is envisioned that this will follow an implementation according to the blueprint in Fig 11 from sub-section 3.4. A flowchart showing the algorithm for the design is shown in Fig 12:

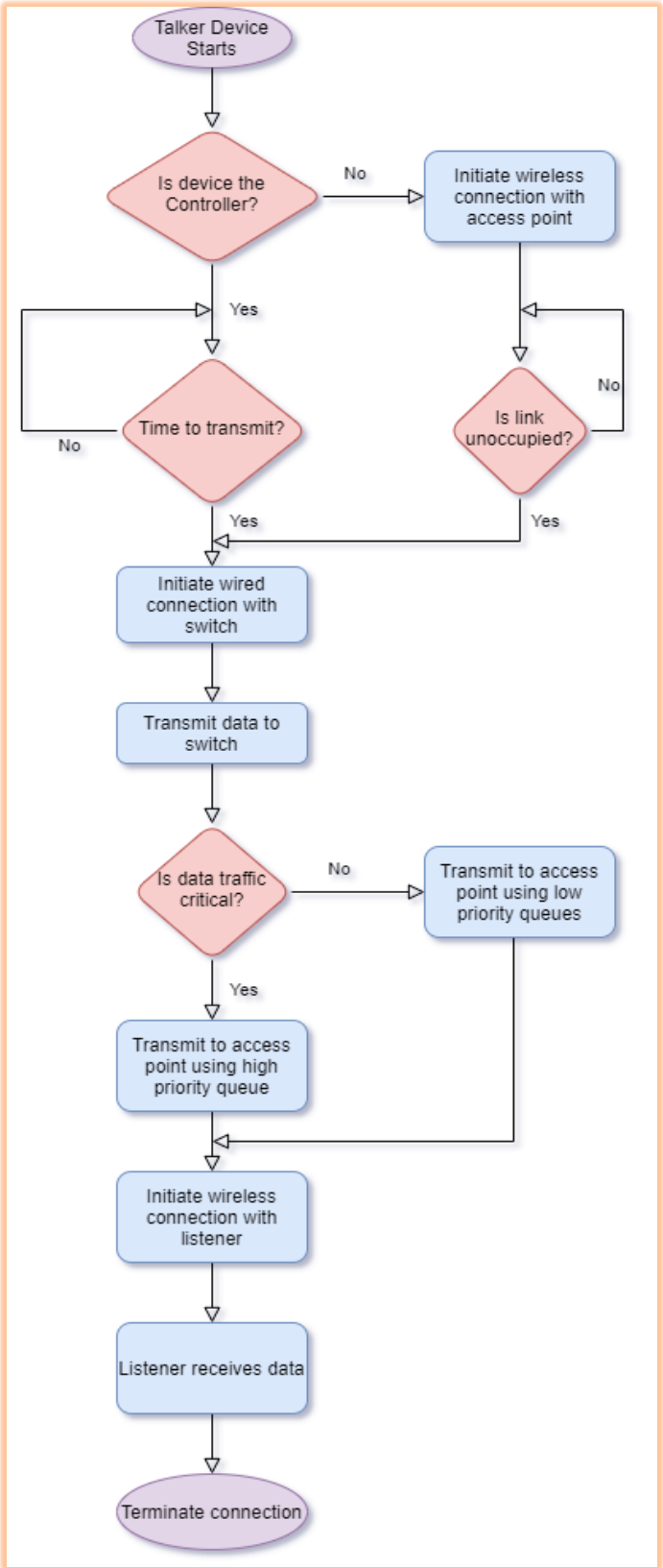

Fig 12: TSN Wireless Operation

There are two groups of end-devices: talker devices and listener devices. Among the talkers, some are designated control devices for urgent traffic, while others have less urgent traffic to send. The goal is for the talker devices to send a message to their target listener devices. For this, they hail the use of a mix of access points and TSN bridges. Before communication can begin, the wireless devices need to join the network connection of the access points; they need to know how to talk to each other.

Control traffic is sent through hard-wired links to the bridge or switch devices, while the background traffic is sent to the nearest access point, which processes the traffic flow based on the order they are received. The time-sensitive messages arrive at the bridge(s) at the assigned time through one interface, while the background traffic arrives through another, which is a wireless interface. The wired interface operates without interference from any wireless transmissions and has a dedicated bandwidth which it does not share with other forms of media (it is less prone to interference and other factors that may affect communication). Control traffic is sent out to a TSN bridge on schedule, arrives on schedule, and leaves on schedule (Fig 13).

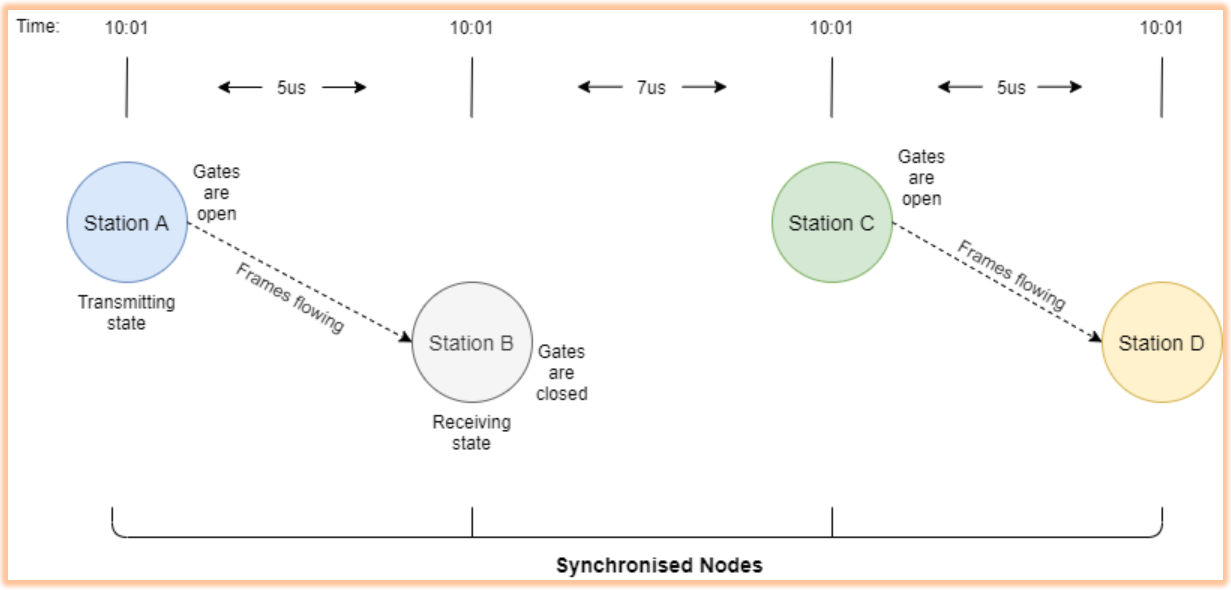

Fig 13: Stations on Schedule

The nodes will be configured thus: When the wireless (non-critical application) talker hosts want to transmit, they signal the wireless access point. The access point changes its state from idle to receiving. The wireless traffic is sent out and arrives at the access point. The wireless traffic will only be allowed to continue on to the switch if the link that connects it to its destination is unoccupied by more urgent traffic. If it arrives to find that the link that is busy, it will have to wait at the switch until such a time the link is free. The switch initiates a connection with the next switch and transmits to it following a schedule. After traversing all

switches, it signals a second access point to ready itself to receive data. It then forwards the TSN frames to the next wireless access point. Once the urgent traffic has been propagated, the non-critical frames can be forwarded. The access point changes its state to transmitting, and it establishes a connection with the intended listener device before it sends out the frames on the now-unoccupied link. Once done, and the listener device concludes reception, it changes its state back to show it is free to accept or coordinate other transmissions. Because the first switch is now free, the access point with the non-TSN frames will send out the frames in the same manner from switch to switch, and finally, from the end access point to the listener device. The listener device receives the message on its wireless radios and terminates the connection upon completion. The switch and access point conclude transmission.

3.6 Chapter Summary

The offerings of this chapter include an explanation on the choice of wireless technology for wireless TSN, and an insight into what stands in the way of conceptualising it over this wireless technology.

There are many common factors that plague any Wi-Fi communication; first and foremost is because it is conducted in unlicensed frequency ranges, which all Wi-Fi and many other wireless technology enabled devices must share. This section of the work also went into the requirements for the proposed solution, and what is needed to support it in terms of the non-wireline medium and the physical or hardware components.

The design considerations introduced a methodology that involved a phased approach to the solution. More to the point, an understanding and consideration of the 7-Layer OSI model was employed because the design stages required traversing across these layers as the idea of transmitting a message through Wi-Fi was conceptualised. Many of the literature (Section 2) concerned with wireless TSN hint at the redesign of hardware, or introduction of new features that contradict the current mode of operation of wireless systems, in order to realise the most complete solution for TSN in a non-wired medium. Since this would involve firstly a thorough consideration of all design aspects by the standards bodies and organisations before producing the new components, the writer's work would need to follow another methodology. It was decided that software reconfiguration to command the state-of-the-art hardware components, as they are, is the most practical and viable approach research-wise.

The chapter that follows details the practical implementations of the wireless TSN solution, using the requirements and design stages introduced in this section to guide the implementation process. The architecture put forth can then be tested out in the proposed environment.

4. Implementation of the Wi-Fi TSN Simulation Model

The previous chapter developed the thinking behind and elicited the requirements for deploying a wireless TSN. It went on to present the conceptualised design with the reasoning behind its perceived architecture. This chapter presents the realisation of the proposed TSN solution.

4.1 Motivation for the Choice of Platform

For this work, the OMNeT++ platform was chosen. There are other simulation programs available; the most notable being NS-2 (and -3 is now available), and the Riverbed Modeler (more popularly known as OPNET[9]). Many others exist, but these are often more specialised in their function, and at the time of this review, there weren't any encountered that were able to support TSN yet. Unlike OMNeT++, the two do not readily work with all OS platforms: OPNET works with Windows and Linux OS; NS-2 works with MacOS, Solaris and Linux, though on Windows, it needs to be installed through the Cygwin software.

NS is short for Network Simulator. Like OMNeT++, NS-2 has a modular design, and it is fully public-source. It is written partly in C++, like OMNeT++, and partly in OTcl, the object-oriented version of Tcl (tool command language). According to [72], among NS-2's main drawbacks, its object-oriented design limits its scalability for WSNs (Wireless Sensor Networks). Also, it implements unrealistic parameters, that is, in [72], parameters used in real life sensor networks are completely different from those in the program. Xian et al. [73] have noted how, although it has many protocol models, they are centred around TCP/IP, which they believe makes these models excessively unitary. NS-3, on the other hand, is seemingly a combination of NS-2 and other simulation tools [72]. The name, NS-2, leads many to believe it is the predecessor to NS-3, but this could not be further from the truth. NS-3 is not backwards compatible with NS-2, and [72] reports that it offers more realistic models too. They also distinguish how it is written wholly in C++, with optional Python bindings, and offers the choice of coding in either C++ or Python (not OTcl like the former). Like NS-2, it also needs third-party software, Cygwin, to be installed on Windows. Both NS- programs are free and open-source and so invite community development for the extension of their models.

Similar to OMNeT++, OPNET is structured into models, whose programs are written and compiled in C++. The review in [74] gives an in-depth analysis on OPNET and its suitability for different use cases: in the lab curriculum, for undergraduate and graduate level students; for the research case, firstly because of its feature-rich library, and next to this, its extensive documentation, which is good for development, testing and validation; in industry, to model real-life situations. In regard to its suitability in industrial communities, the analysis also revealed that its library of existing and already implemented components and models means

[9] The OPNET modeler and OPNET Technologies in its entirety has since been acquired by Riverbed Technologies, so it is not easily found by the same name, although the name OPNET is still widely used and understood.

that development does not have to begin from the ground up. This leaves less room for errors and supports reliability; the simulator can be reliably used for testing and development in commercial and mission-critical applications [74]. The main drawback comes, however, in its closed nature – its source code does not provide access to its simulation kernel [75]. Developers cannot easily develop in it or debug certain problems in their simulations using it. In [76], Pahlevan and Obermaisser do not have their TSN implementation, which is based on OPNET, publicly available. The networking community cannot add to, use or extend it [23]. Moreover, if work is not open-source it is not easy to compare it against similar applications and tools for benchmarking or other purposes. Falk et al. [23] stated that they could not compare the OPNET TSN implementation in [76] to their own TSN simulator (NeSTiNg) with regards to its performance and how accurate it is.

Additionally, OMNeT++ is fully public- or open-source and widely extensible. It encourages researchers and developers to add to its frameworks and leverage its underlying power and capabilities. Varga and Hornig [75] credit this as being the reason for its plentiful simulation models and frameworks.

Some studies have OMNeT++ as faring better than OPNET and NS-2; according to [73], OMNeT++ is the more powerful simulator as attested to by results from a WSN simulation experiment they conducted with these three, based on criteria of run time, delivery rate, and memory requirement. That along with the greater number of supported functionalities, such as the powerful simulation library, and debugging capabilities, made it come out on top. But because this work isn't a comparative study of simulation frameworks, it will not go more in depth than this into what makes OMNeT++ better than other tools. This work is rather one more concerned with the tool best used for the simulation of TSN.

Finally, OMNeT++ and all the underlying tools chosen for this particular work can be freely obtained over the internet because they are all open-source.

Added Motivation

Aside from the capabilities mentioned in the previous sub-section, another major driving factor for choosing to use OMNeT++ to implement the wireless TSN solution is for its NeSTiNg framework. This framework was the only tool found, whose authors openly made its protocols and examples available to be tested out and experimented upon. Many tools that the writer came upon were not so. Of the notable works that exist, two implementations stand out: one based on the CoRE4INET framework; another is actualised using OPNET. Jiang et al. [22] were able to leverage the CoRE4INET framework to develop their TSN simulation model by employing gate control lists (GCLs). They used these GCLs to enable their switches to filter through time-triggered traffic (critical traffic) and other types like interference traffic and best-effort traffic. The TSN features they primarily realised are IEEE Standard 802.1AS for

synchronisation and IEEE Standard 802.1Qbv for scheduling. The development of their model and the evaluation of their obtained results is detailed in their work. However, they did not release their source code for public eyes. CoRE4INET also extends the INET framework (to be presented in Section 4.2), and so it runs atop INET and OMNeT++, as does the proposed model in this work. Pahlevan and Obermaisser [76] developed an IEEE Standard 802.1Qbv-capable switch model which also supported the sub-protocol IEEE Standard 802.1Qci titled Per-Stream Filtering and Policing. In order to implement these two time-based TSN mechanisms, they used the OPNET simulator, which, just like OMNeT++ is a discrete event-based platform. It is important to note that [76] as well do not have their work publicly available. Therefore, it cannot be used or extended by the wider community of researchers and developers.

For those authors that were more inclined to show their work, the problem is that their tools are not conformant with the IEEE 802.1 TSN standards. Heise et al. [77], developed their tool called TSimNet, whose entire source code and documentation they made available. The TSimNet model is another TSN simulation framework that is an extension of the INET framework, again built to run on OMNeT++ as its simulation platform. Its authors managed to implement the IEEE Standard 802.1CB titled FRER, and, among the three main TSN features discussed in this work, they implemented IEEE Standard 802.1Qbu for frame pre-emption. Their work features none of the time-based mechanisms of TSN [23], [77], mainly, synchronisation and scheduling. The disadvantage with this, as they discovered in their results, is that frame pre-emption alone does not always help produce reduced latency times, it is very much dependent on how the system is configured. That is why their whole work was open-sourced; this they did in hopes to encourage developers to implement missing features for their model.

In summary, there are therefore two main reasons why NeSTiNg was chosen as a basis for this work: firstly, for its overall completeness in implementing TSN, and secondly, for its open-sourced nature. The NeSTiNg simulator has implemented the three main features of TSN: IEEE Standards 802.1AS, 802.1Qbv, and 802.1Qbu. By making their work open, Falk et al. [23] allowed for the extension of the framework used for their TSN implementation in many ways, such as those described in this very work which ultimately helped in efforts to deliver the proposed solution. Suffice it to say, as evidenced from the prior text, there aren't many tools that have taken both above factors into consideration.

As an extension framework it has the advantage of directly benefitting from the many capabilities of the modules in the underlying framework(s) that it is built to extend. The NeSTiNg simulator will run alongside the INET framework, for one, because it means to borrow from its packages to implement its own models. It thus calls for the INET framework to be installed along with it. This is described next.

4.2 Overview of the Implementation

The writer envisions simulating a TSN environment with a model suitable for addressing the earlier listed objectives (Section 1.5), and for simulating the proposed system design in Section 3.4. As per the previous section of work, Section 4.1, there are two tools that look to fit this purpose: OMNeT++, and NeSTiNg underlying it. Motivation on why they were chosen has been provided in that section too. OMNeT++ comprises a library built on extensible, modular and open-source principles – it is a discrete event simulator [78]. The Network Simulator for Time-Sensitive Networking (stylised as NeSTiNg) also handles discrete event simulation, coming off the fact that it is based on OMNeT++. It started as an internal tool for TSN evaluation [79] and it can be used to analyse the behaviour of converged IEEE 802.1 TSN networks, as evidenced in the work of Falk et al. [23]. In this way it is more specialised in its function. NeSTiNg was initially developed by a group of students at the University of Stuttgart. It is currently maintained and extended by the Distributed Systems group of IPVS, at the same University, who have left instructions supporting its use on their Github page [80].

Since the set-up involves a wireless environment different from the traditional case of wired Ethernet, it calls for the need to leverage other capabilities of the existing networking tools. The INET framework is an OMNeT++ library for wired, wireless and mobile networks; it contains models for the Internet stack, wired and wireless link layer protocols [81]. Since it has extensible support for the wireless use case, it is intended to be employed as a building block for the set-up.

To illustrate this environment, Fig 14 shows a diagram similar to the one in [22, Fig. 1], but with a modification.

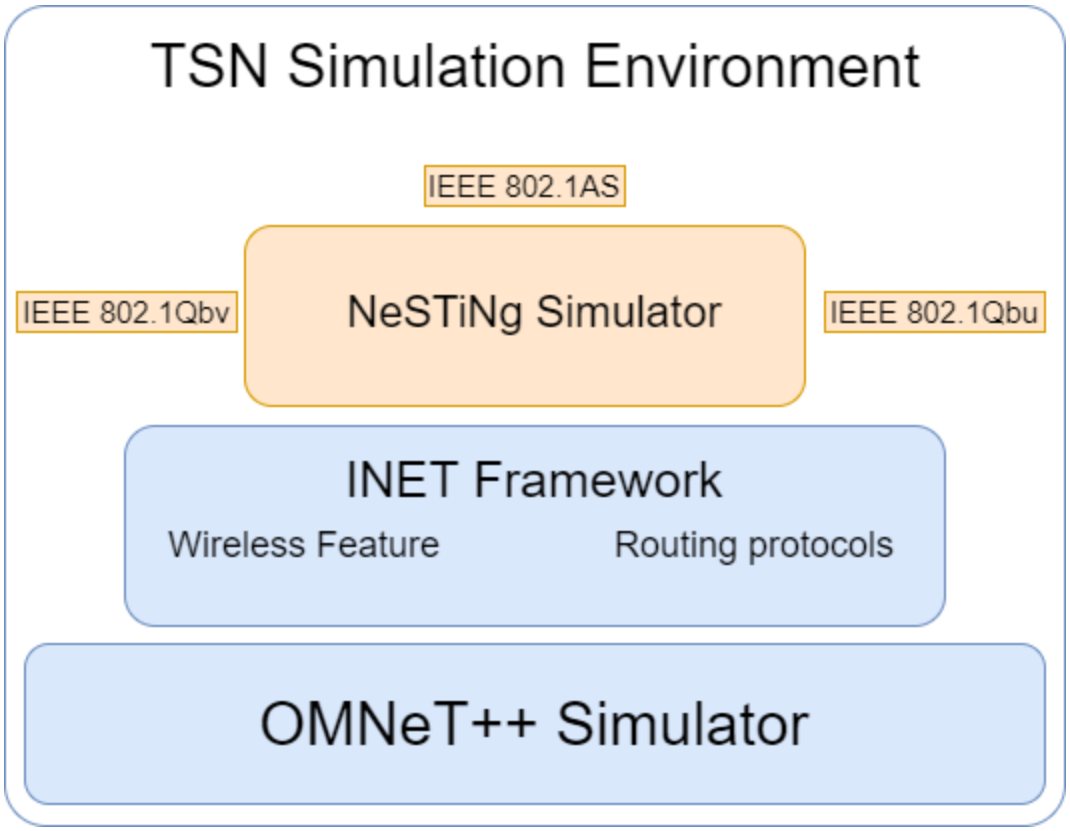

Fig 14: TSN Simulation Environment

The writer will use NeSTiNg primarily to build the base of the environment. Then he will draw from the underlying INET framework, calling on its models with hopes to extend the wired capabilities of NeSTiNg TSN into the wireless domain. After the network devices have been built and it is verified that the devices can communicate with one another, OMNeT++ will be used to stage the simulation of the network. The proposed system will seek to model the most primary TSN features, that is, timely synchronisation (IEEE Standard 802.1AS), its timed gating or scheduling mechanism (IEEE Standard 802.1Qbv), and frame pre-emption (IEEE Standard 802.1Qbu).

To help answer the research questions and meet the objectives raised in Section 1, this chapter will comprise the following stages:

1) reviewing the specific components of the identified simulation environment, and identifying their specifications and requirements (Section 4.3);
2) presenting some initial assumptions for, and limitations of the framework environment (Section 4.4); afterwards,
3) it will get into the implementation of the design using the mentioned framework tools and software (Section 4.5); before ending with

4) a description of the test environment that will be used to help evaluate its performance (Section 4.6).

4.3 Components of the TSN Environment

In this section, all the components of the TSN environment are identified. These include the hosting software's specifications and other tools identified from the methodology in order to produce the requirements and specification needed for the design stage. On reaching out to the authors of NeSTiNg, it was discovered that NeSTiNg has no set minimum system requirements; it rather depends on the size and configuration of the scenarios.

Hardware

In the writer's simulation, the following device is intended to be used:

- A Windows 10 Enterprise host machine with the following specifications:
  - Intel Core i5 processor, 3.2 GHz
  - 64-bit OS (operating system), x64-based processor
  - 8 GB RAM (Random Access Memory)

Software

NeSTiNg is only supported by Linux at present, and so a Linux OS distribution to host the system environment was used. The lab's machines run Windows OS, it either had to be replaced or to work alongside a Linux OS. The latter was chosen, for reasons explained in detail in Section 4.5.1. In order to accurately model the deterministic network, the following virtualisation software are a major requisite:

- Oracle VM VirtualBox ver6.1.2
- A Linux guest / Virtual Machine with the following specifications:
  - Ubuntu-18.04.3-desktop-amd64 for the ISO / disc image file
  - 64-bit OS
  - 2 GB RAM, with 50 GB dynamically allocated storage / hard disk file

The wireless solution itself comprises three key technologies. A brief review of each will follow in the sub-sections to come, along with a motivation for their choice. In each case, the latest version at the time of writing that was able to support NeSTiNg as evidenced by its authors was used.

- INET ver4.1.0
- OMNeT++ ver5.4.1

4.3.1 NeSTiNg

NeStiNg is built to work in Linux OS and so does not at present support other systems such as Windows. It may be able to work with the Windows Subsystem for Linux (WSL), but this

hasn't been tested extensively yet. Typical installations for other systems apart from Linux rely on a virtual machine to host a guest Linux OS.

4.3.1.1 NeSTiNg Architecture

The NeSTiNg simulator is a tool that extends the INET framework to incorporate TSN-specific features with the introduction of certain simulation models [23]. Among these features, it implements scheduling mechanisms IEEE Standard 802.1Qbv for scheduled gating through a time-aware shaper (TAS), IEEE Standard 802.1Qbu and IEEE Standard 802.3br to aid in frame preemption, and IEEE Std 802.1Qav as a credit-based shaper (CBS). It can be used to build different TSN environments and test how they behave. The writer purposes to use it to realise the proposed wireless configuration and analyse the behaviour of the resultant TSN network.

To provide the forementioned features, the tool's authors built certain technical attributes into their framework as seen in [23]. The original version of the framework they used in compiling these initial findings has since changed. This work is based on the more current version of NeSTiNg which was tested with the INET and OMNet versions mentioned in the software specifications section. In light of this, the models and illustrations in this writing reflect the technical attributes of this current version. Furthermore, the writing is also based on the modes of operation of this same version, which were observed and studied. This version was found to differ somewhat from the original earlier version in terms of things like the structure of some directories and the environment's visual design. The technical attributes will now be presented, and the operation sequence detailed.

To acquaint the TSN environment with different classes of traffic, NeSTiNg uses **frame tagging** firstly, to support it in differentiating between traffic with contrasting QoS profiles or requirements. It tags frames as per the IEEE Standard 802.1Q for supporting VLANs on an IEEE Standard 802.3 Ethernet network. The VLAN tag is composed of a 3-bit identifier known as the Priority Code Point (PCP), which associates frames with a particular queue. Up to eight queues can be established per outgoing or egress port. Although the exact way that traffic in these different classes is treated is left to the implementation, the IEEE made some broad recommendations since the release of their IEEE Standard 802.1Q-2005 standards documentation. It has since been superseded, with the latest iteration being IEEE Standard 802.1Q-2018, which still maintains the original definitions, redrawn from [82] in Table 4. The NeSTiNg authors do not explicitly mention which particular version they use, although in their paper [23] they refer to the 2018 revised standards. This is also what will be used in the proposed implementation.

Table 4: Traffic Type Acronyms [82]

Priority	Acronym	Traffic types
1	BK	Background
0 (default)	BE	Best effort
2	EE	Excellent effort
3	CA	Critical applications
4	VI	"Video", < 100 ms latency and jitter
5	VO	"Voice", < 10 ms latency and jitter
6	IC	Internetwork control
7	NC	Network control

With increasing number of queues, more traffic types can be supported with a distinct level of priority. PCP that indicates a frame priority level of 1 is the lowest priority, while priority 7 is the highest. It is easy to draw parallels between this PCP identifier and the one detailed in the fourth question of the review portion in Section 2.4, as they are similarly used to prioritise flows.

The industry-grade TSN-enabled Cisco switch mentioned there also has a unique identifier that comprises three variables: the destination MAC address, VLAN, and the CoS (Class of Service). Peculiarly enough, this CoS is also expressed in three bits like the VLAN tag employed in NeSTiNg. And it is this very CoS found as a field in the Ethernet frame's header that groups different traffics into different classes of varying levels of service priorities, much like how QoS differentiates between different traffic. For some scholars, the similarities of CoS with QoS end in the traffic management that they provide [83]. CoS does not guarantee the level of service, in contrast to QoS, which provides guarantees on the bandwidth and latency. CoS only offers a best effort level of service. Further, it can be said that CoS functions as a form of QoS that is limited to layer 2 in standard Ethernet, whereas QoS operates on layer 3 of the OSI model.

In NeSTing, another significant addition is in the **TSN-capable switch**, its components and how it processes frames. A look inside this switch device (Fig 15) will reveal, among other things: the *relayUnit*, the *processingDelay* component, and the Ethernet module. The Ethernet module is part of INET's framework and is supported by the NeSTiNg switch. The switch below has four Ethernet interfaces. A frame that arrives at a switch enters through one interface

before it is sent up to the *relayUnit*. Here, using information from the *filteringDatabase* and *interfaceTable,* if needed, it is routed to the *processingDelay* component for a simulated delay before it exits through an egress port on one of the other Ethernet interfaces. A global clock keeps time to record the duration of these events and the simulation. This is the general set of steps which a frame follows on entering and leaving a switch in NeSTiNg.

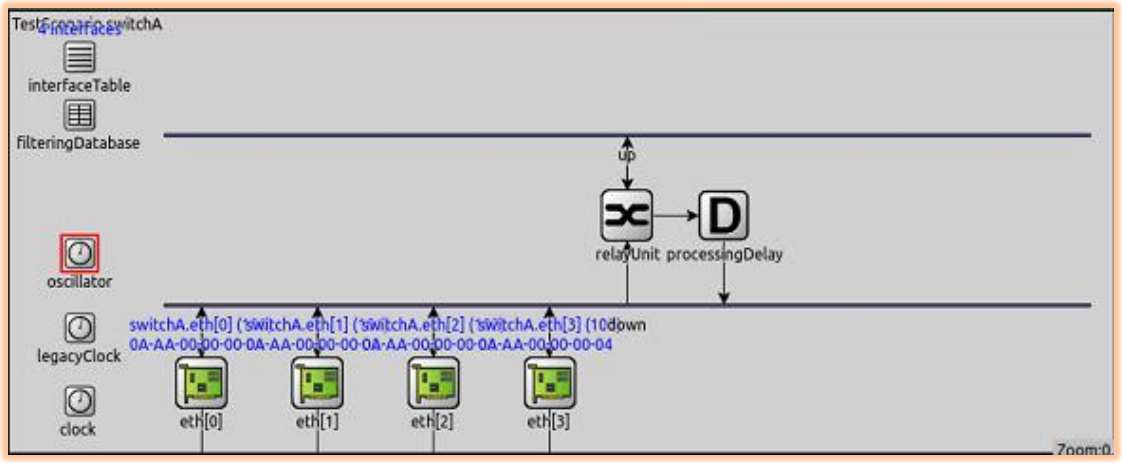

Fig 15: Inside NeSTiNg's TSN Switch

Further peering a level deeper, inside the Ethernet interface device (Fig 16) shows: the *encap* components, the *queue* component, and the MAC component. Here the frame gets encapsulated before being requested by the *queue* component, and it is placed in a queue. It is then sent to the MAC layer which models transmission delay. The MAC component sends out a signal before the frame can be transmitted on the link. The reverse order happens for an incoming frame when the port is used for ingress, except it bypasses the *queue* component. It heads straight to the *qencap*, then the encap components, before making its way out the Ethernet module, and restarting the procedure at the *relayUnit* (where, again, any necessary information is gathered before it can make its exit on an egress port, as explained previously).

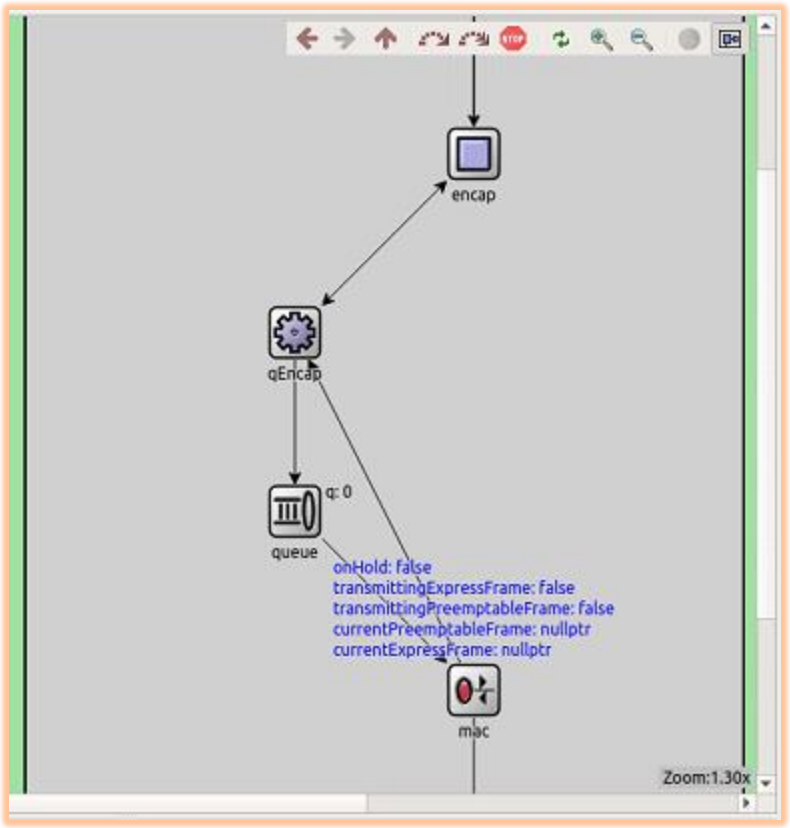

Fig 16: Inside an Ethernet Interface – NeSTiNg TSN Switch

Further technical developments are seen in the scheduling mechanisms, implemented by logical devices known as **TSN shaper components**. The TSN standards talk of two shaper components: the time-aware shaper (see IEEE Standard 802.1Qbv in [9]), and the credit-based shaper (see IEEE Standard 802.1Qav in [84]). These can be observed by going yet deeper into the mechanics of the *queue* component, shown in Fig 17. This comprises: the *queuingFrames* component; the queues, eight of them numbered from zero to seven; the *tsAlgorithms*, one for each queue, the *tGates*, and finally; the *transmissionSelection* component. The overall mode of operation is set to follow the "strict priority" mechanism, although NeSTiNg also provides two other options: "gating", and "frame pre-emption". A packet is sent down from the *queuingFrames* component to one of the various queues. The *gateController* component coordinates the *queuingFrames* component to set the gates on or off using some binary value. The operation of a traffic light/robot is a good analogy for the mode of operation of this *queuingFrame* component: it dictates what queues should stop, how long they wait, and which queue can go, or "release traffic".

Initially, all gates are in closed state, that is, "11111111". After it is requested, the *queuingFrames* component places the frame into the queues according to the value in its PCP field, earlier explained. In "strict priority" mode, only frames that are in high priority queues

will be transmitted. Supposing two events are scheduled concurrently, that is, for the same simulation time, and a frame of high priority arrives after one of lower priority; ordinarily it may have to wait for the first packet to be transmitted before itself. This would be the case were it not for the shaper components, namely the tsAlgorithms[q] and the tGates[q] components, which represent the CBS and TAS respectively (having defined a set of maximum queues as Q, "q" refers to the index of the specific queue involved, which could be any number from 0 to 7). The events in these shaper components are thus processed **before** the transmission selection is performed. Take, for example, a scenario where two packets had arrived at the *queuingFrames* component, then these two are queued up, say in queues six and five, in that order. Queue six happens to be the express queue; the message placed in queue six is there because it is of a higher priority as designated by the value in the frame's PCP field. The next body of text concerns the sequence of events that follow.

The *queues[6]* component starts off by signalling the *tsAlgorithms[6]* component with a *packetEnqueued()* message, and likewise, the *queues[5]* component signals the *tsAlgorithms[5]*. These in turn signal *tGates[6]* and *tGates[5]*, also in that order: *tsAlgorithms[6]* sends a *packetEnqueued()* signal to the *tGates[6]*, and immediately after, the *tsAlgorithms[5]* does the same with the *tGates[5]*. This action is repeated when the *tGates[6]* signals the *transmissionSelection* component. Finally, this component sends out the *packetEnqueued()* signal outside this *queue* component to the MAC component. This concludes the first stage.

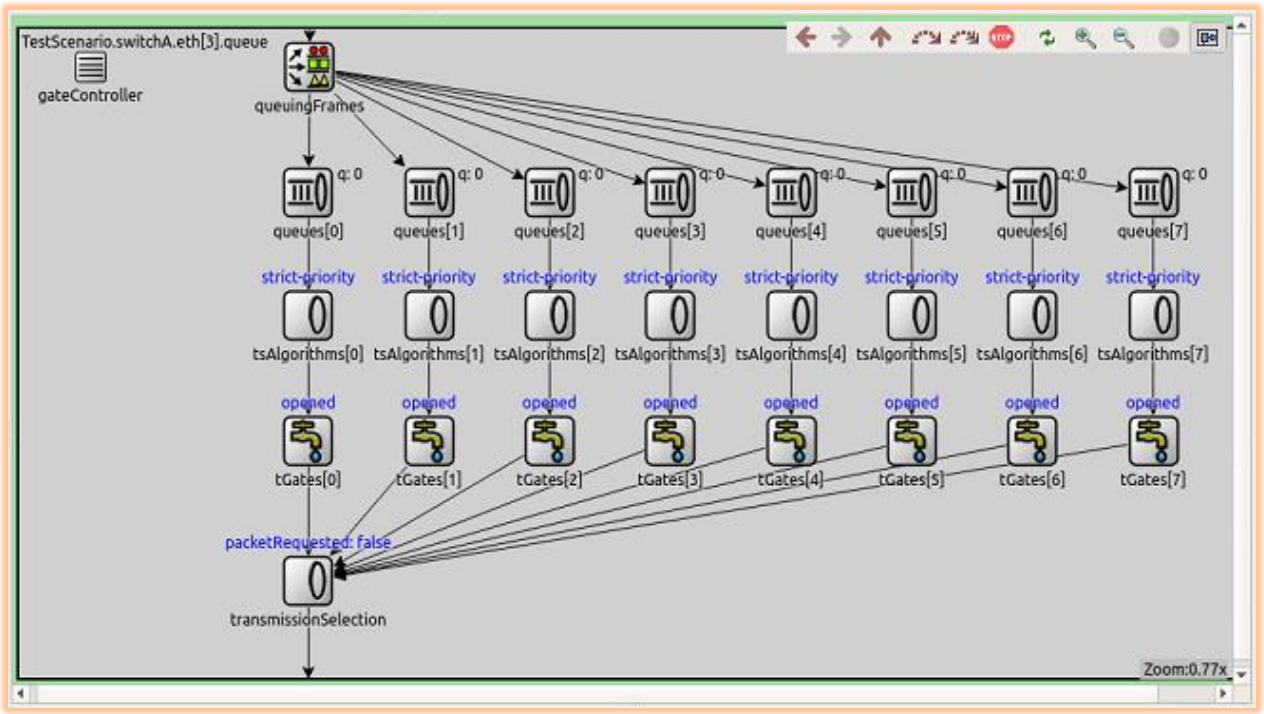

Fig 17: Inside the Queuing Component – NeSTiNg TSN Switch

The next stage is initiated when the MAC component sends out a *requestPacket()* signal into the *queue* component, specifically to the *transmissionSelection* component. This sets off a reverse chain of events. Assuming the two message queues from the start were in contention of transmission, *queues[6]* mentioned as being the express queue, the *transmissionSelection* component sends out a *requestPacket()* only to *tGates[6]*, consequently "selecting" *queues[6]*'s data stream, as its name suggests. The same signal, albeit with a slight change, now *requestPacket(maxBits)*, gets sent next to *tsAlgorithms[6]* (this passes the maximum bits value into the *requestPacket()* method of that interface). It, in turn, signals *queues[6]* with the *requestPacket(maxBits)* message in the same manner. Now *queues[6]* has the go-ahead to release the message and so sends it down the same path that the *requestPacket()* signal came from. Once it reaches the *transmissionSelection* component, it ends its journey in this device, and ultimately concludes this stage by proceeding on to the MAC component for transmission. It subsequently leaves the MAC component as an Ethernet signal. When the next frames arrive at the *queuingFrames* component, the process is repeated.

In Fig 18 [23, Fig. 1], all the main technical features developed in the NeSTiNg tool detailed herein can be seen at work. It shows the inner workings of the egress port of a TSN-ready switch during the time interval with duration ti_2. There are eight queues of traffic ranging from priority zero to seven. They are modelled by the CBS before they reach the gating controlled by the TAS. The TAS, as has been explained, determines which queue gets to transmit at what time by the opening or closing of its gates. Only queues whose gates are in open state can transmit. In the case above, this is the queue whose PCP is priority 6. The CBS is currently shaping the traffic in queues 5 and 6.

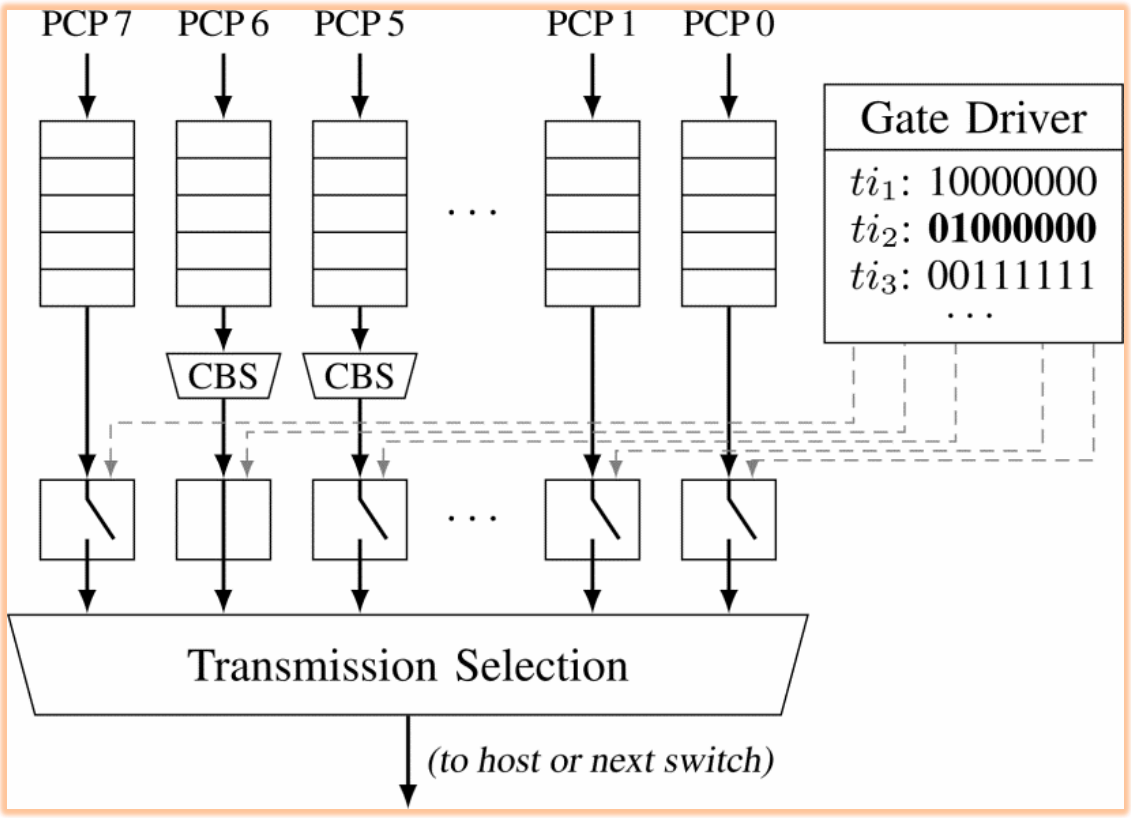

Fig 18: TSN Transmission Selection Architecture [23, Fig. 1]

4.3.2 INET

The latest prescribed version as reported on the NeSTiNg site that the current NeSTiNg simulation framework has been tested with is INET ver4.1.2, on a Linux system. However, at the time of conception of this work, the available stable version which was tested with the framework was ver4.1.0.

INET has been used in a few other time-sensitive communications simulations [22], [77]. These were reviewed in an earlier sub-section: Section 4.1.1. It can either be used on its own, or alongside another key framework that may extend it. It is worth mentioning that since INET is an OMNeT-based framework, all these implementations are staged on top of it. INET can be used to design and validate new protocols, and is thus suited for experimentation [81].

The INET API is structured in this manner; it comprises a number of packages that house the separate modules, module interfaces and channels. These sub-components may inherit from or extend each other and are in this wise further divided into simpler components. For example, a compound module is implemented by a simple one which inherits from it. These modules may further be divided into sub-classes, and they consist of sets of properties, parameters and other definitions that allow one to attach details to a particular component. These details can be in form of character or integer variables, for example, and they allow one

to describe or articulate the properties of the component. By the use of this modular structure, INET's features support a wide variety of communication networks, and can, for the purpose of understanding, be categorised into the following broad categories: wired networks, wireless networks, mobile networks, ad hoc networks and sensor networks. It further supports popular models like those for the Internet stack of protocols, including IPv4, IPv6, TCP, UDP; those for link layer protocols, like Ethernet and IEEE 802.11 Wi-Fi; and MAC sensor protocols, among others. Fig 19 shows the writer's INET directory which has been expanded to reveal its contents. It has folders that contain models for the different equipment and protocols that make up the OSI layers, such as models for routing in the network layer, and models for describing the node components. It also has other models that exist strictly to enhance the simulation experience, such as the visualizer folder, whose files help describe the visual settings of the simulation platform. An example of this could be to display link lines and make visible the communication range of transmissions.

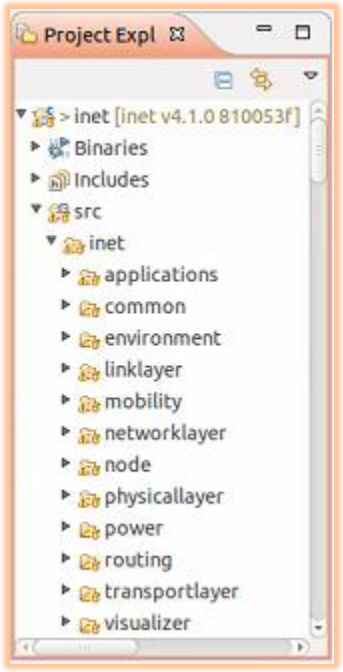

Fig 19: INET Directory Structure

The writer sought to use INET since it can openly be extended and used to prototype new layouts and protocols such as those called for in this work. It will allow for the reuse of features for the envisioned model, and this can be validated in a simulation setup using the OMNeT++ platform described next.

4.3.3 OMNeT++

OMNeT++ has a simulation kernel that is written in C++, which renders it compatible with all platforms that have a C++ compiler. It will work on Windows, Linux or MacOS. In fact, OMNeT++ is the abbreviation for Objective Modular Network Testbed in C++.

4.3.3.1 OMNeT++ Overview and Architecture

The latest version listed by the NeSTiNg authors that they have tested the current version of their tool with is OMNeT++ ver5.5.1, run in a Linux environment. But at the time of writing, the author only had access to ver5.4.1 as the latest recommended version for the framework.

OMNeT's authors tried to make it as general as possible to support a range of communication networks [75], [78]. It is powerful enough to be used by academics, educational and research-oriented commercial institutions to simulate computer networks, distributed or parallel systems [75]. It achieves specialisation through the frameworks that it supports. As such, it doesn't directly provide simulation devices to generate computer networks, but rather it provides the tools and capabilities through which these frameworks design these simulation components. Apart from INET and NeSTiNg, it even provides support for mobile technologies by the addition of frameworks like SimuLTE [85], which extends INET in order to simulate and evaluate the performance of LTE networks.

Along with the modular architecture that allows for the addition and extension of frameworks, as mentioned, OMNeT++ has a GUI (Graphical User Interface) that is populated through the use of NED files. NED, short for *NEtwork Description*, can be understood as OMNeT's topology description language, through which the structure of models can be defined [75]. One can choose to either write up code, or to work solely in the GUI. Components are assembled to produce complete networks in these NED files. OMNeT++ models are composed of compound modules and simple modules, their interconnections and the gates controlling the flow of data. It is worth noting that the NED language has an equivalent eXtensible Markup Language (XML) representation to which NED files can be interconverted without the loss of data or even the comments made in the original files [75]; it is seen to be quite versatile.

After networks are built using modules and components in the NED files, these components can be attributed further properties and module definitions to define their behaviours in the same file. But the configuration of settings along with the definition of values for these variables and properties is done in INI files. One NED file can be reused by several INI files each specifying its own different combination of parameter values for the same network components. This is a convenient way of providing multiple simulation scenarios or implementations for the same base network setup.

In Fig 20 the interface of the OMNeT++ Integrated Development Environment (IDE) can be seen.

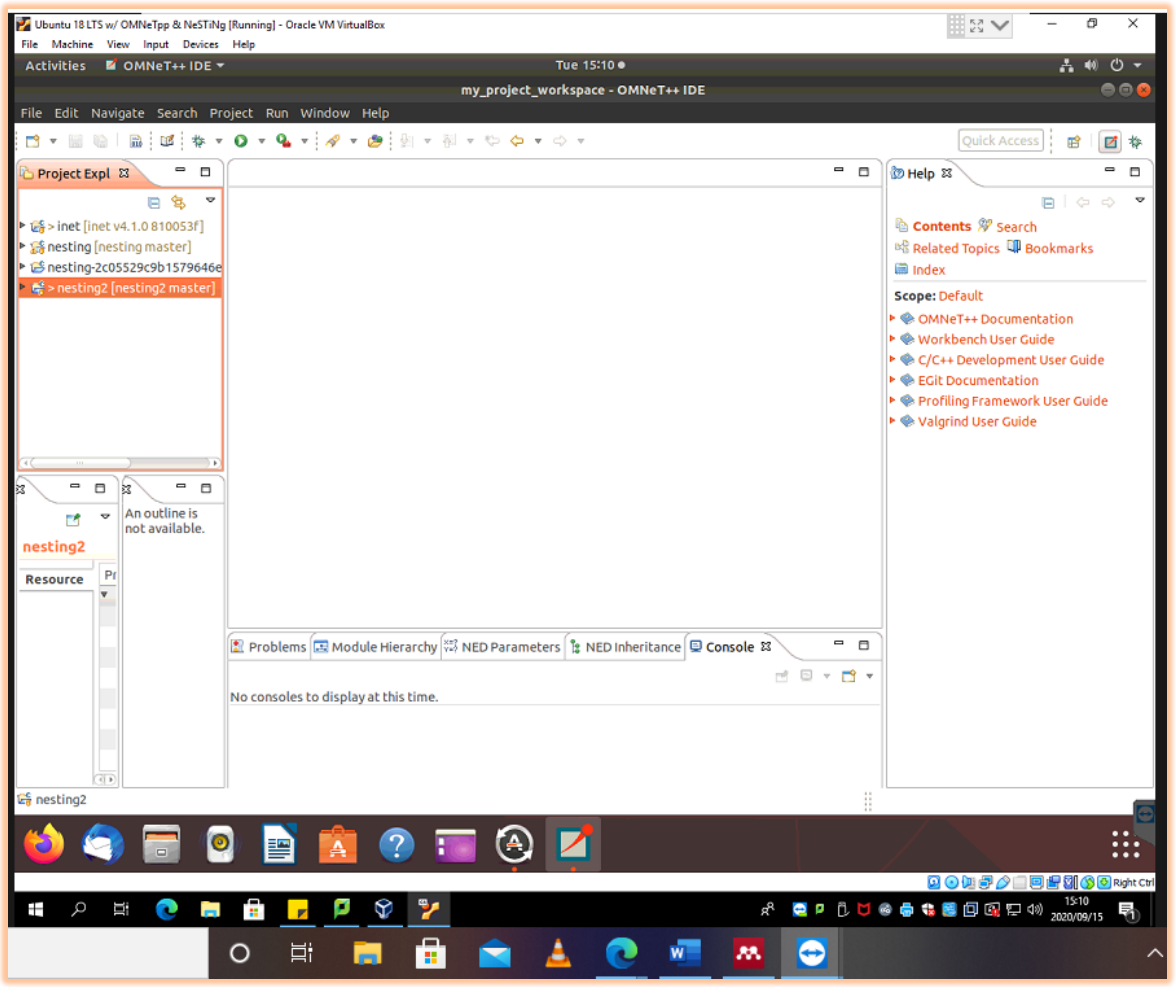

Fig 20: OMNeT++ IDE Workspace

On the left side of the console screen, the directories listed correspond to the frameworks employed: INET and NeSTiNg. It should be noted that, aside from the main branch, other versions (Github branches) of NeSTiNg were used to compare the performance of the developed proposed system in different testing environments.

4.4 Initial Assumptions and Limitations

Before going about the design, some initial assumptions were made to work with in the design process, and to specify certain foreseen limitations of the proposed system design.

4.4.1 Performance in Shared Frequencies

Some initial assumptions can be made based on the limitations of the present standard of Wi-Fi: it operates in license-exempt frequencies and factors such as network congestion curtail its performance. Some sources believe this is the singular reason why even future versions of Wi-Fi may not be able to guarantee fully deterministic communications [34]. As a result, the assumption is that the design probably may not be as efficient or as optimal as the

conventional wired implementation. This work explores in part if these limitations can be mitigated, through its introducing wired functions to be integrated with the wireless network. It focuses more on making use of Wi-Fi in its current state for time-sensitive communications.

4.4.2 Concerns of Erroneous Results from Memory Limitations

The system's hardware specifications were an initial point of concern; the host system had an internal memory of 8 GB of RAM. Of this, only 2 GB could be safely allotted to the virtual operating system without having a detrimental effect on the performance of the host system that ran the virtual machine. This places a limit on how resource-intensive the virtual machine simulation program can be. However, if the amount of resources required at any one time exceeds a certain pre-set value, the system begs use of "swap memory" to extend the available system resources. A more detailed explanation of this is provided in Appendix A3.

The memory proved sufficient for some initial verification tests, and it was concluded that it could always be extended if the system became too burdened or sluggish to cope in latter stages. Moreover, the experimentation could be repeated to get some reliable average value so as to defeat any bias or erroneous results spawned from isolated incidents.

4.4.3 Lack of Individual Sync Feature

Another factor that is foreseen to limit the simulation of state-of-the-art TSN concerns the synchronisation of the different network elements. Although it won't necessarily limit the throughput and timeliness of the setup, it will not be able to fully mimic conventional TSN. In real-world implementations of TSN, it is expected that each switch or bridge maintains its own clock. The BMCA algorithm is used to keep these individual clocks synchronised according to IEEE 1588. This is not the case with the NeSTiNg tool, as will now be explained. A global clock is used for all active elements on the simulation canvas.

It would make sense that the first aspect of TSN to be incorporated into the model is synchronisation, mostly because other mechanisms such as scheduling and frame pre-emption are built on the basis of one understanding of time. However, research revealed that the NeSTiNg simulator currently runs switches with local clocks as opposed to having one global or synchronised time. The writer personally reached out to the authors of the NeSTiNg tool and confirmed their plans to introduce further aspects of TSN, including clock synchronisation protocols, only in future versions [23]. Since NeSTiNg does not have a sync feature for even the normal wired TSN, the writer decided to forego implementing a global clock for the feature as it would require introducing a feature that implements the BMCA algorithm and IEEE 1588 PTP protocol, to be integrated into all other modules from the base – something that would detract the attention away from the main focus of realising a wireless model. Instead, the writer focused on porting other features such as strict-priority scheduling into the wireless domain. The global clock could be taken up as an area for future research. The author did although work to include, as a possible basis for this future work, an algorithm

that shows the breakdown of how the 1588 PTP sync mechanism could be designed into the framework. This is included in Appendix B.

4.5 Implementation of the Wireless TSN Design

This stage focuses on using the requirements specified from an earlier juncture to produce a model of the proposed system as it operates in its simulation environment. The aim is to incorporate IoT principles of reliability into the completed design so that it simulates a model that holds reliably well even in mission-critical scenarios.

Before anything else, the TSN platform was set up with the application software and tools needed for the task at hand. This meant the installation of the components identified in Section 4.3. What follows is a summary of these first actions.

4.5.1 Configuring the TSN Environment

NeSTiNg is only supported by Linux at present, and so it deems the use of one of the Linux OS distributions. Since the lab machines come pre-installed with Windows OS, it either had to be replaced or to work alongside a Linux OS. Four options existed for this: the first, and least desirable, would be to erase the current profile and reinstall Linux as the host OS for the machine; to install Linux alongside Windows in a dual-boot setup; another is to use a fairly recent development in Windows that allows it to run Linux as a service, termed Windows Subsystem for Linux (WSL); finally, to incorporate virtualisation in order to run Linux as a guest OS inside the Windows host device. All these options are described in Appendix A1, along with the reasons for why the writer went with what they did.

The writer chose the last option; virtualisation was used to configure this work's NeSTiNg environment.

Then the component tools could be configured. As the abstract illustration in Fig 14 (Section 4.2) shows, this began with the OMNeT++ discrete event simulator as the staging tool for the model, then the INET framework of programs to allow for the inheritance and reuse of features and models, and finally, NeSTiNg ran ontop of all as per the recommendations in [80]. By virtue of this, it was able to extend features of the underlying INET framework.

A detailed install procedure of the above with complete steps for each action is provided in Appendix A2.

Appendix A2 also contains a list of common errors that were encountered in each sub-stage of the installation phase as well as how they were resolved. Many of them are a result of unmatched program dependencies usually stemming from the fact that the programs offered in the Ubuntu repository are outdated. When these prerequisite programs go missing, it leads to errors of all kinds. Most others are due to errors in the code. Aside from explaining the fixes, links to the sources that may have helped debug the issues are also cited and provided.

4.5.2 Development of the Wireless TSN Simulation Model

Once the TSN environment was set up, the actual construction of the proposed model could begin. To this end, the original NeSTiNg repository with all its folders and files was replicated into the newly created wireless TSN project, called *nesting2*, and programs development began directly in these files.

The illustration in Fig 21 depicts a typical workflow of the stages a simulation model undergoes from when it is first created to when it can be deployed or ran in the simulator.

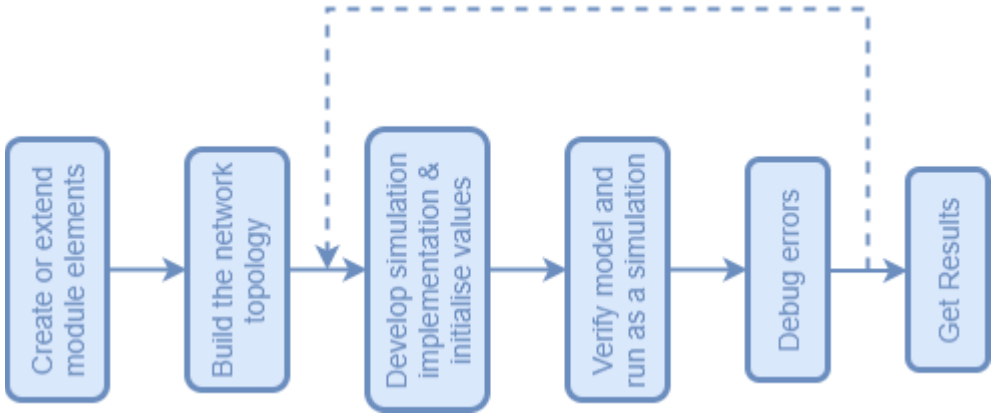

Fig 21: Simulation Model Workflow - from Development to Deployment

When building a network, one has the option to use modules in the way they come without further enhancements. But for the purpose of this work, it behoved the author to first **enhance the already existing modules** since they were not in their current state able to achieve the aims that the study catered towards, for example, the inclusion of wireless input and output gate definitions. Thus, the first stage consisted of creating or extending the module elements – inheriting what properties were there and applying modifications where it was needed.

The next stage is synonymous with building the topology of the network with NED files, OMNeT++'s topology description language. These can be populated using either the GUI, XML files or through program coding; the latter style was preferred throughout the development of this work (including other stages) because of the writer's conversancy with programming. The NED files allow one to define the structure of models, as was cited earlier in Section 4.3.3.1. The writer was able to define their network using the earlier inherited and modified module components. The assignment of simple attributes can be done in the same NED files. But the proper configuration of settings and the initialisation of values and parameters is done in the INI files, in the stage that concerns the development of simulation scenarios or implementations. It is pluralised because it is understood that the same base network setup can be reused by multiple simulation implementations, where each one has the desired combination of parameter values.

At the close of this stage the INI file implementation could finally be verified by the system and run as a simulation using the OMNeT++ discrete event simulator.

Debugging is an optional stage to correct the code if the simulation picked up any errors in the simulation model. May it be noted that there is need to differentiate between errors and warnings. The former constitutes bugs that need to be removed. Some bugs result in warnings that are of low-impact and need not be resolved; those due to the use of legacy parameters in the modules, for example. These parameters are a valid part of the framework, but for some reason the system tracks them as non-existent.

The procedural algorithms that show what more went into the actual development of the programs used in the first three stages are described next. They followed the logic behind the design sequence detailed in the methodology in Section 3.4.

Sequence 1 Create or extend module elements

Inputs:

- Ethernet node element
- Non-critical frames from traffic generator application

Outputs:

- Wireless node element
- Non-critical frames

Development Environment:

- NED language
- NeSTiNg, INET, OMNeT++ bundle of programs
- Ubuntu 18.04 LTS (Long-Term Support)
- OMNET++ IDE System Editor

1: Open existing node for modification or create one
2: Import relevant packages and modules
3: Define and initialise network node parameters
4: Define gate for the wireless interface
5: Inherit required submodules and define their behaviours, example wireless interface submodule: specify parameters for encapsulation, MAC layer values, wireless radio
7: Define network connections for traffic flow, example input-output, interface upper and lower-layers
8: Error-check the program
9: Upload program to source code repository

Sequence 2 Build the network topology

Inputs:

- Wired implementation topology
- XML files

Outputs:

- Wireless network setup

Development Environment:

- NED language
- NeSTiNg, INET, OMNeT++ bundle of programs
- Ubuntu 18.04 LTS
- OMNET++ IDE System Editor

1: Open wireline topology for modification
2: Import required packages and modules
3: Initialise topology parameters
4: Define or inherit topology types, example the channel
5: Build the network with submodules (including the newly-created one) and configure their parameters
6: Assign network connections for the different elements and nodes
7: Verify the network and connections
8: Upload program into code repository

Sequence 3 Develop simulation implementation and initialise values

Inputs:

- Wired TSN model or implementation
- base network setup or topology

Outputs:

- Wireless TSN simulation model

Development Environment:

- INI configuration file
- NeSTiNg, INET, OMNeT++ bundle of programs
- Ubuntu 18.04 LTS
- OMNET++ IDE System Editor

1: Import network topology/setup and XML files
2: Initialise simulator settings, for example: simulation time limit, results directory
3: Initialise values for model components, for example: host addresses, packet length, Wi-Fi band attributes

4: Configure settings for TSN traffic flow, model components, and simulator canvas
5: Run the simulation
6: Debug simulation if error code is returned
7: Upload source code to repository

Section 4.8 directs to the source code repository containing the programs that the detailed sequences describe, and all the code that supports this work. In brief, Appendix A2:

- lists the main files used in the development phase
- specifies other program files which were created and used for testing purposes and to aid with debugging
- further lists the main files which were reviewed because they were required for understanding of the inner workings of the framework models

Additionally, the system stored the results generated from the various tests and simulations in their respective results folders, also provided in the appendix.

May it be noted that, apart from procedural instruction, Appendix A also documents the errors that were encountered during the installation and development phases of this work. These errors might be common to other prospective or future developers on OMNeT++ and NeSTiNg. For this reason, the appendix file has been uploaded onto the code repository, too, in case it may be able to aid any others in their research or academic work. It is seen in the main parent folder, as the README.md file directs.

4.6 Description of the Experiment

The substantial precedents that formed the configuration and development phases in the former sections laid the groundwork for the following experiment. Originally, [23] used the following network node devices:

- *robotController*
- *roboticArm*
- *workstation1*
- *workstation2*
- *backupServer*
- *switchA*
- *switchB*

Their ([23]) model was initialised in the writer's environment, reproduced here in Fig 22 to help describe their setup.

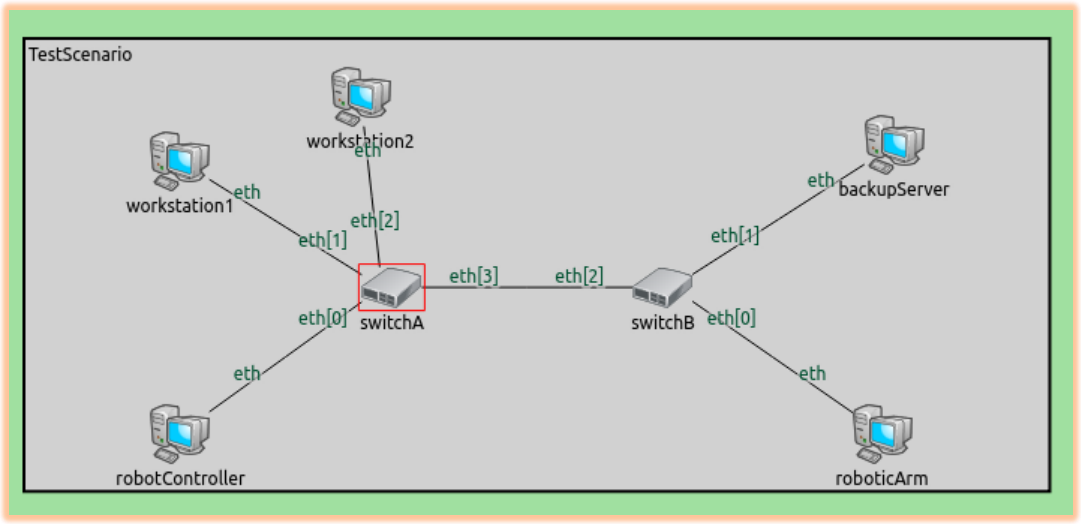

Fig 22: NeSTiNg TSN

For this work, the writer had to modify these initial pieces, as well as add a few more, for example:

- *accessPoint1*
- *accessPoint2*

In said modification, the *workstation*, *roboticArm* and *backupServer* host components were configured to use wireless capabilities that supported the existing network traffic. The access points do not readily support TSN.

The premise remained the same. The architecture is that of a converged network, with time-sensitive traffic and non-critical traffic accommodating the same infrastructure. There are two distinct network traffic flows:

1. A traffic generator originating within the workstations (talker devices) tries to occupy the network with background traffic whose destination is the *backupServer* (listener device).
2. Amidst these ongoing transmissions, the *robotController* (talker) with critical traffic is transmitting to the *roboticArm* (listener) to control its movements.

The initial wireless implementation developed is shown in Fig 23.

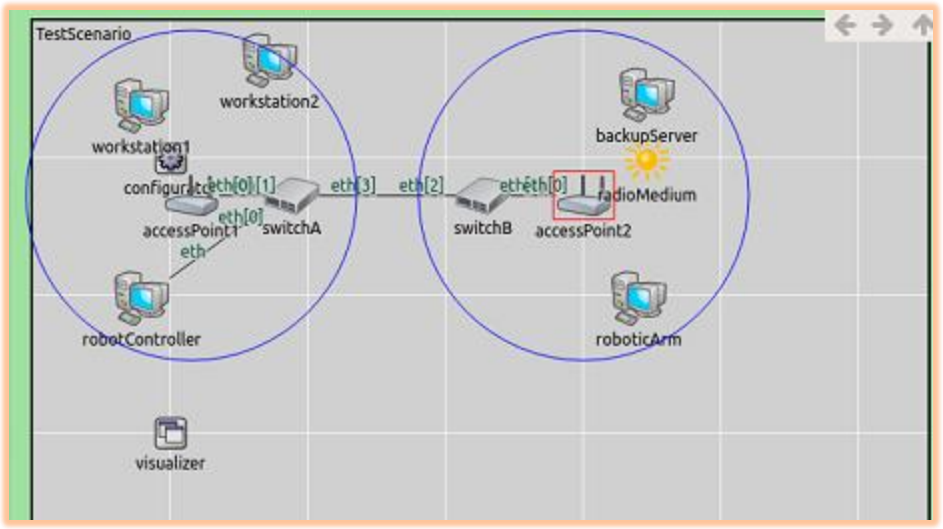

Fig 23: Initial Wireless Implementation

What changes is that, now, the workstations are transmitting through the access points instead of just the switches. So, the initial best effort flow changed from purely cabled communication to one that integrates wireless radios for host and receiver. As for the control or time-triggered traffic stream, it still went to the switch initially, and then it was transferred to a wireless signal directed towards the wireless *roboticArm* listener device. Additionally, the wireless devices were first "associated" with the access point that was meant to service them, so that the particular host could be managed by it. The circles show the communication ranges of *accessPoints* 1 and 2.

Some of the main parameters the author's system worked with are shown in Table 5.

Table 5: Main Environment Parameters

	Parameter	**Value**
	Wi-Fi 5 (802.11ac)	
1	*Constant speed propagation*	$2.99792 \times 10^8\ m/s$
2	*Centre frequency*	$5\ GHz\ (20\ MHz)$ *(at kTB = -101 dBm)*
3	*Modulation*	*BPSK*
4	*Radio transmitter power*	$100\ mW$
5	*Radio receiver sensitivity*	$-85\ dBm$
6	*Radio receiver energy detection threshold*	$-85\ dBm$
7	*Radio receiver* SNIR *Threshold*	$4\ dB$
8	*Radio medium background noise power*	$-110\ dBm$
9	*Ethernet LAN datarate; delay*	$1\ Gbps\ (highest);\ 0.1\ us$
10	*Wireless LAN bitrate*	$693.3\ Mbps\ (highest)$

Others not mentioned here include those dealing with the wireless contention window, queues and antennas.

Falk et al. [23] tested the efficiency of their TSN implementation for the scenario having the switch configuration of "strict-priority scheduling". Each TSN flow had eight queues, working with the same size frames and each of equal highest priority. Non-TSN frames were placed in lower priority queues. This work's setup will also have its components configured similarly, including having strict-priority scheduling.

4.7 Chapter Summary

This chapter identified and presented the components necessary for the development of a wireless TSN model as per the requirements from earlier stages. The TSN environment consists of three main elements, namely, the OMNeT++ discrete event simulator tool, the INET framework of programs, and the specialised NeSTiNg framework. With the introduction of these tools, a justification for their use was also provided.

The chapter also made some initial assumptions to work with and examined some foreseen limitations for the final product. First of these was that evidence from scholarly journals suggest that it will not be as optimal as the conventional wired design as it operates in shared frequencies, although its integration with the Ethernet media is expected to help mitigate these performance-hindering factors. Nextly, slight variations are expected in the values of results owing to memory limitations. Finally, that it will not be able to mimic the exact process for the synchronisation of individual elements, as the simulation only has a global clock sync feature to work with. In light of that last limitation, the chapter's offerings also included an algorithm detailing the flow of events for the introduction of the individual elements sync feature, as a possible basis for future work.

The actual implementation included a configuration of the TSN environment, and the focused development of the wireless model. All the tools sourced for the implementation can be freely obtained from their relevant authors. Likewise, the code used in the development of the proposed model has been openly hosted, with provisions made for its reuse, as specified in the next sub-section. Subsequent to the programming exercise in the development phase, an experiment to compare the two models, conventional and wireless, was introduced.

In the chapter to come, the wireless implementation will be benchmarked against the wireline one. The results produced from the tested models will be analysed to investigate whether possibly they may speak or shed light on the future of Wi-Fi TSN.

4.8 Source Code

All the source code used in the development of the solution proposed in this work is hosted on Github, a popular source code management and versioning control web application. It can

be freely accessed and downloaded from the following link: https://github.com/ArnoKinabo/NeSTiNg-Wireless-TSN.

5. System Evaluation

With the help of some tests and simulations, an evaluation of the system and its performance is conducted. The results of the evaluation will be analysed to see how they speak to the research questions and earlier requirements.

5.1 Wireless TSN Simulation

Several experiments were run using some reduced models for the proposed system shown previously in Fig 23, with aims of benchmarking it against the original, fully-wired model provided by [23] in Fig 22, and getting a better understanding of the possible ranges of expected results.

Experiments of three base types were conducted: an isolated best-effort (or non-TSN) traffic flow (Fig 24); another solely with critical (or TSN) traffic (Fig 25); and, lastly, a combined setup of both TSN and non-TSN traffic (Fig 26).

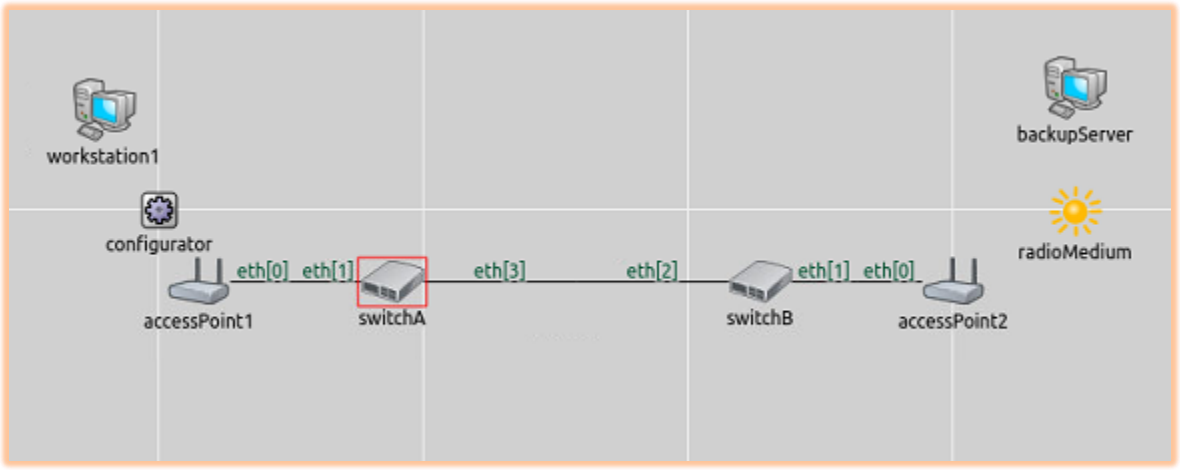

Fig 24: Best Effort Traffic Only Setup

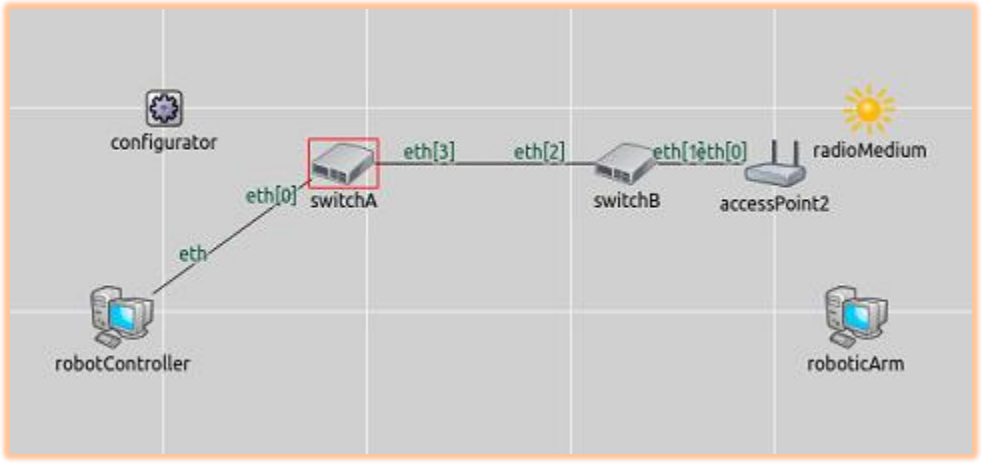

Fig 25: Critical Traffic Only Setup

Fig 26: Best Effort + Critical Traffic Setup

In the first setup, a single non-TSN talker element, *workstation1*, communicated non-critical traffic to a non-TSN listener, the *backupServer*. In the second setup, the TSN talker, *robotController*, sends TSN frames to the TSN listener, the *roboticArm*. And in the last, both types of talkers send their different traffics to their respective listeners. A traffic generator was used in the workstation to send out a steady stream of 1500-byte frames at an interval of 12 µs. In the controller, the cycle time was defined as 400 µs for the 354-byte TSN frames.

5.1.1 Wireless Simulation with Two Access Points

The number of frames which were successfully delivered in all three setups was recorded for nine cases whose raw data is in Table 11 (Appendix C), and ultimately the throughput performance was calculated using Eq 2.

$$Throughput = \frac{bytes}{packet} \times \frac{bits}{byte} \times \frac{1}{\frac{seconds\ (time\ interval)}{packet}} \div (1024 \times 1024)$$

Eq 2

The cases differed in that each had a specific Ethernet and wireless speed pair attached to it. In other words, three simulations were run for each case, each corresponding to a different setup type.

The raw data for this and all further tests can be found in Appendix C. They will be identified plainly as needed, along with the tables they correspond to. Beginning with this Table 11, its summary is rendered here in Table 6. As for Fig 27, it pertains to the data from Table 11

(Appendix C). It presents the results obtained from the writer's analysis of this first set of experiments.

Table 6: Throughput (Mbps) Summary for experiments with setups shown in Fig 24, Fig 25, Fig 26

	Wireless speed in Mbps (Wired speed in Mbps)	non-TSN alone	TSN alone	TSN w nonTSN	nonTSN w TSN	TSN+nonTSN
	11 (10 Eth)	2.03	2.59	2.32	1.67	3.99
	36 (10 Eth)	5.28	5.29	6.74	1.93	8.67
	36 (100 Eth)	4.77	5.29	3.25	12.22	15.48
	54 (100 Eth)	6.20	5.59	3.00	16.26	19.26
	693.3 (100 Eth)	9.42	6.75	1.94	25.15	27.09

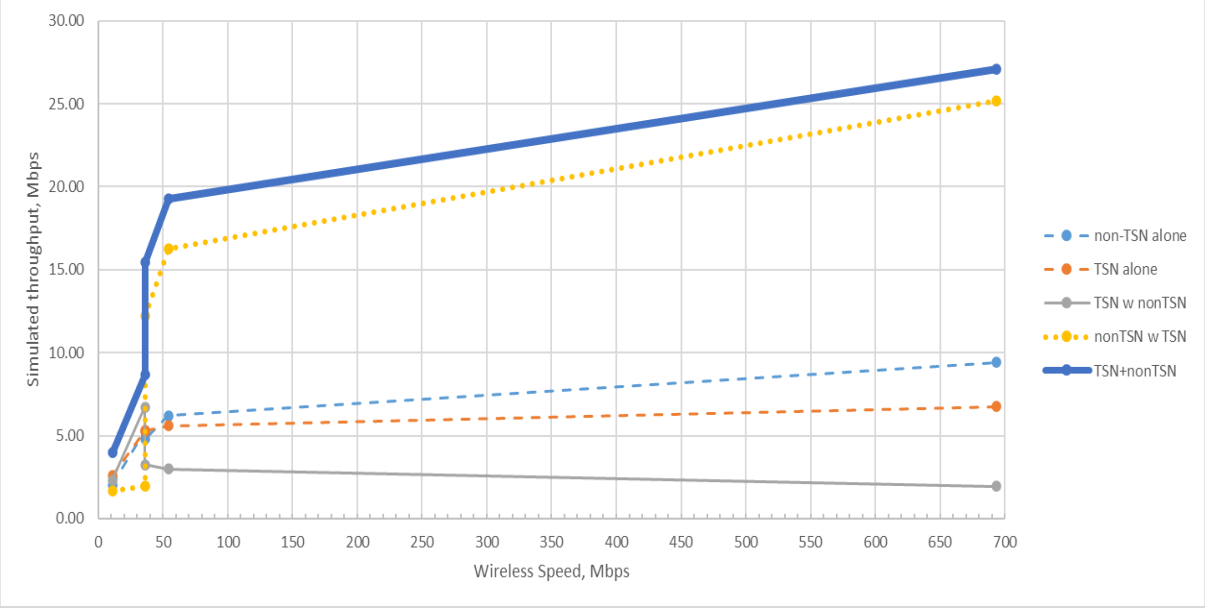

Fig 27: Throughput vs Speed for experiments with setups shown in Fig 24, Fig 25, Fig 26

The legend items can be explained thus:

i. "non-TSN alone" denotes the results for the best-effort flow in the first setup;
ii. "TSN alone" shows the results for the critical traffic flow in the second setup;
iii. "TSN w nonTSN" signifies the results for the critical traffic in the presence of other lesser priority frames in setup three;
iv. "nonTSN w TSN" conversely relates to the results for best-effort frames as they co-exist with critical frames in the same setup three; finally,
v. "TSN+nonTSN" is an amalgamation of the last two traffics in the same speed bracket.

In various cases, the bandwidth was increased from 10 Mbps to 1 Gbps for Ethernet, whereas for wireless, it ranged from 11 Mbps to 693.3 Mbps (this is the actual speed which

corresponds to a theoretical throughput of 1 Gbps), which is the highest that the simulator could provide for 802.11ac which was used in the simulations.

This first run of experiments revealed that there was something severely limiting the flow of both TSN and time-insensitive (non-TSN) traffic. For the given frame send intervals, 83,333 packets were expected to be delivered at the *backupServer*, and 2500 packets at the *roboticArm*. Further calculations using Eq 2 would show that, for the given packet sizes and intervals, this translates to an expected traffic of 953.7 Mbps and 6.8 Mbps, for non-critical and critical traffic respectively. This gives a total of 960.5 Mbps; a figure quite close to the 1 Gbps maximum link speed. But even with the highest link speeds, the attained throughputs were far below the expected mark: 20.57 Mbps and 4.04 Mbps, respectively.

5.1.2 Wireless Simulation with One Access Point

In investigating the cause of the discrepancy between the link speeds and measured traffic, the setup was simplified by removing the first access point, connecting the workstation directly to the switch. The parameters were kept the same, that is, one workstation transmitting 1500-byte packets at 12 µs intervals. Likewise, the parameters for the controller remained unchanged. The results from the raw data (seen in Table 12 and Table 13 of Appendix C) could not be properly interpreted because, for many cases, a queue error resulted. This was a runtime error that informed that excessive traffic exceeding link capacity was being generated. After further tests, the exact minimum intervals were established (Table 14 – Appendix C), below which propagating frames would result in the queue error.

5.1.3 Investigating the Bottleneck in the Wireless Simulation

The next phase of experiments was conducted to pinpoint the problem and understand where the packets were lost and why.

A counter was placed at *switchB*. For an *etherSwitch* element, the module that records the kind of signals for the statistics that are of interest to this work is the ***macRelayUnit***. There are three possible packet statistics that can be obtained:

1. @signal[packetSentToLower]
2. @signal[packetReceivedFromUpper]
3. @signal[packetReceivedFromLower]

The second and third are more useful to this work. When the counter is used for the second item, it will count the packets which a switch sends out, which originate from the higher mac layer and are passed to the lower, before they leave the switch through the egress port. The third item does the opposite, it denotes packets that are incoming, that is, they are received from the lower layer before being sent up. That is why, when the *packetReceivedFromUpper* signal is used at an ingress port like port 2 of *switchB* in the following diagram, the counter there records a total of zero packets received, since there is no traffic flow originating from

listeners to the talkers through *switchB*. Which means there are no outgoing packets on that port, but only incoming ones.

For these reasons, the counters used for the following illustrations only traced signals #2 and #3: *packetReceivedFromUpper* and *packetReceivedFromLower*. The results follow.

5.1.3.1 Investigating for the Fully-wired

Fig 28 shows the NeSTing authors' fully wired setup from [23] which has been slightly modified to have a single workstation.

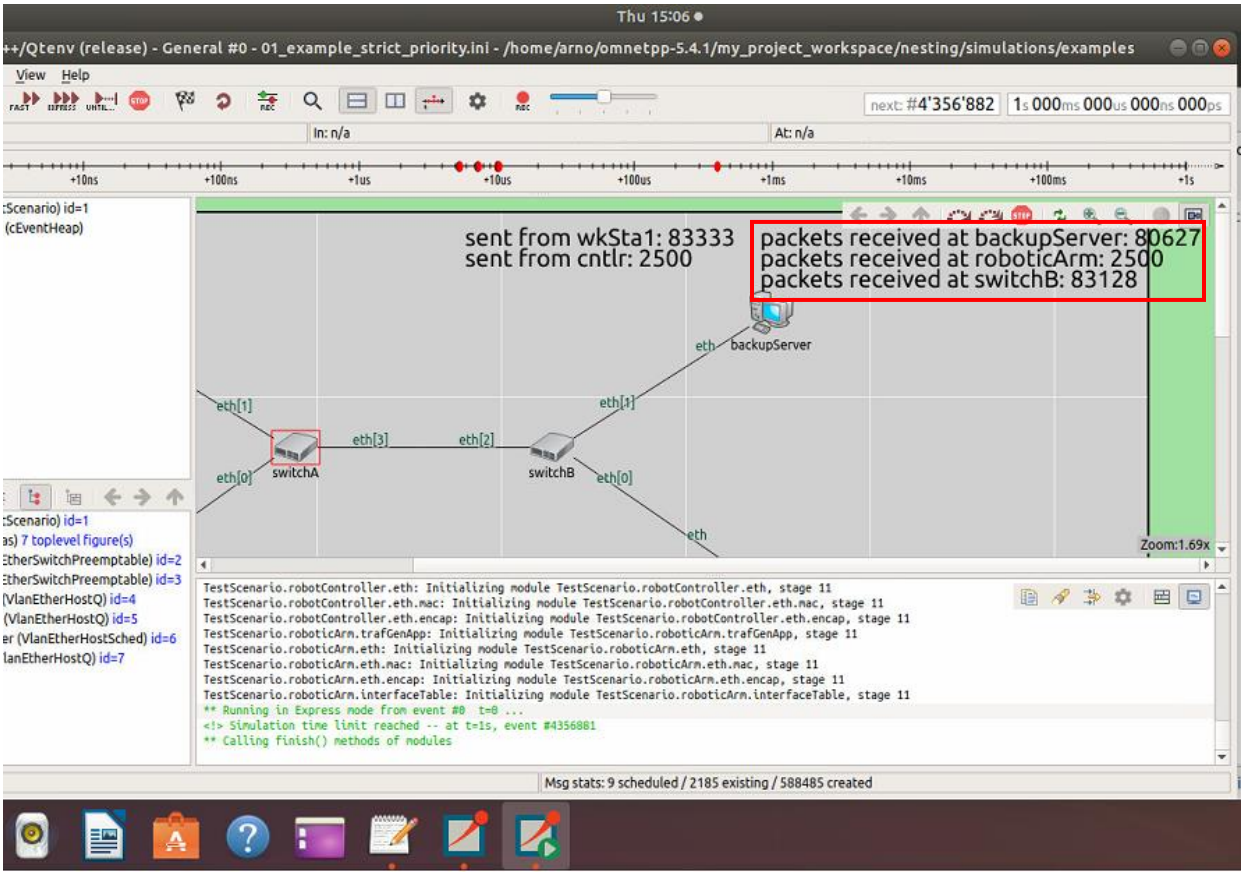

Fig 28: Fully wired - # of packets

For the parameters:

- TSN frame cycle = 400 µs (corresponding to 2500 packets sent in 1 second), packet size 354 B
- non-TSN frame send interval = 12 µs (83,333 total packets), packet size 1500 B
- Eth speed = 1 Gbps

The highlighted section in Fig 28 shows that over 83,000 packets were received at *switchB*, and circa 97 % of the total sent were received at the *backupServer*. All TSN packets were delivered; all packet losses happened in the best-effort traffic. This is the model case relying on wired TSN switches.

5.1.3.2 Investigating for the Wireless with Single Access Point

Next, shown in Fig 29, the semi-wired case with one access point was run, where wireless bandwidth was set to 693.3 Mbps. All other parameters were kept as before.

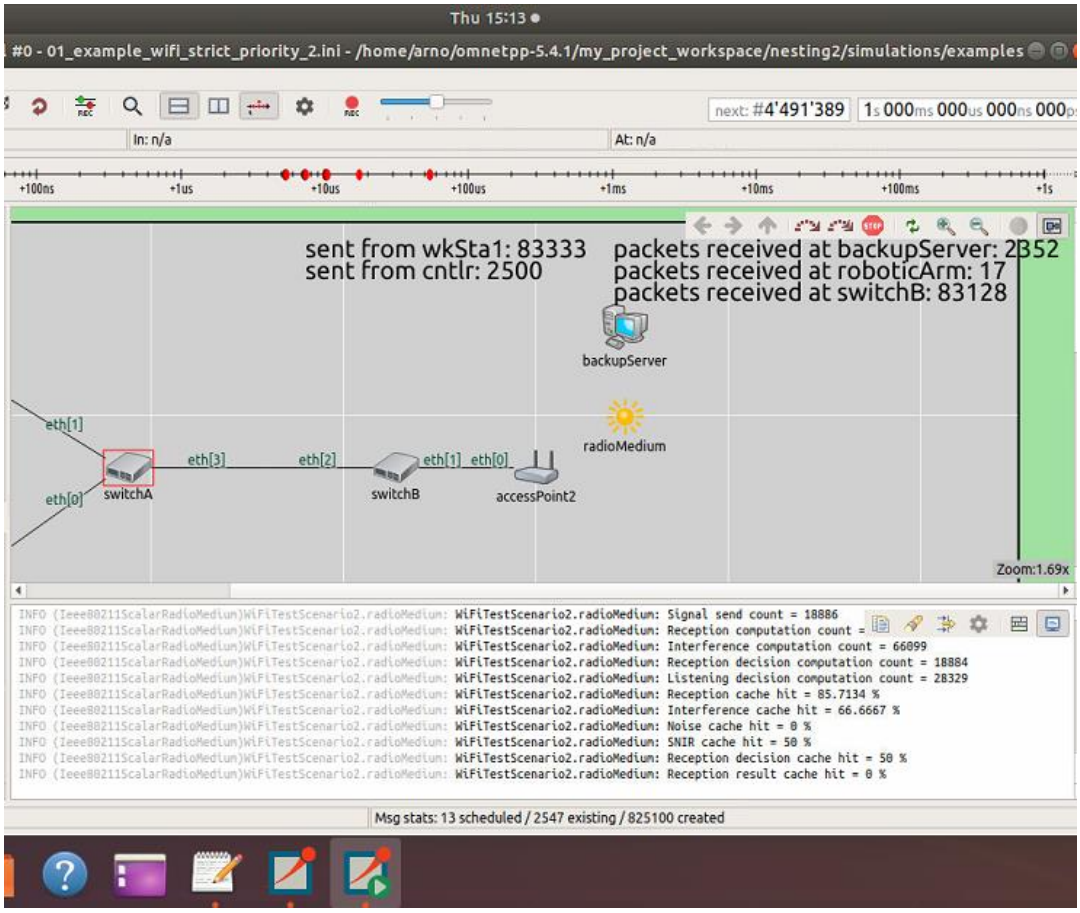

Fig 29: Semi-wired, 1 AP - # of packets

From Fig 29, the full number of packets were seen to be delivered at *switchB*, when counting packets received at interface eth[2]; but most packets seem to have been lost in wireless *accessPoint2*.

Even if one decides to record packets sent out from *switchB* on link interface eth[1], using signal #2, that is, ***packetReceivedFromUpper***, the same number outgoing in the semi-wired setup is observed as is the total for both egress ports in the fully wired setup (2500 + 80,628), which is: 83,128.

Roughly translated,

$$packets\ in\ =\ packets\ out$$

It should be noted that placing a counter at *switchB* was chosen specifically because if the results were similar to that of the fully wired case, where all TSN frames and nearly all non-

TSN frames were delivered, then it would show that all links prior to that point were working optimally. Then in that case, there is no need to measure packet flow again at, say *switchA*.

5.1.3.3 Investigating for the Wireless with Both Access Points

Lastly, the case which uses two access points is presented:

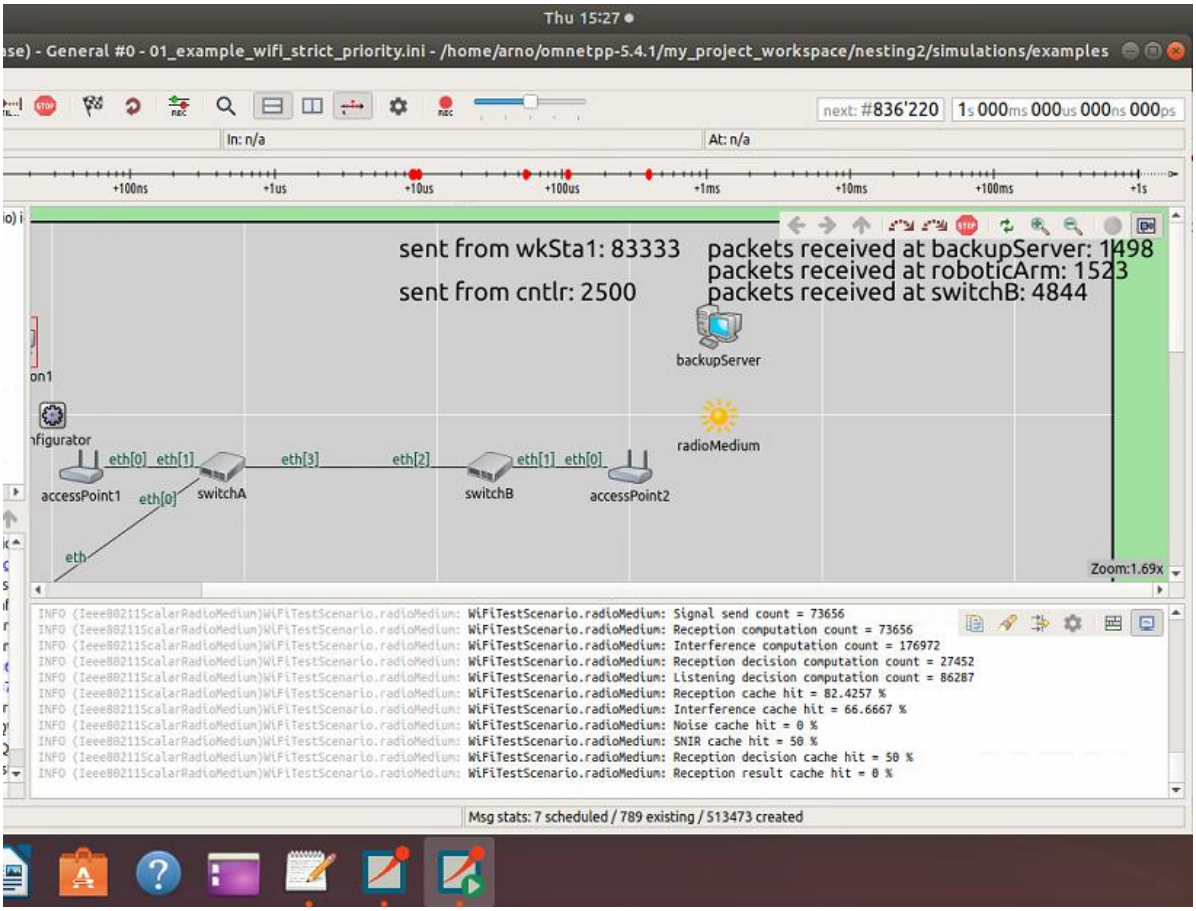

Fig 30: 2 AP's - # of packets

As noted in Fig 30, very few packets arrived at *switchB*, a fraction (~5.6%) of the 85,833 sent out from the talkers. They amount to 4844 packets.

And similarly, on the outgoing interface, eth[1], 4844 packets were counted there too (packets in = packets out).

The only difference between these and the prior two experiments being the inclusion of an access point before the first switch.

From these, it was evident where the problem lay: the majority of the packets in the network were lost due to the access points.

Although it had always been expected for Wi-Fi to underperform given the high rate of collisions in freely accessible wireless mediums, it was not foreseen that the achievable throughput would fall to such low levels.

5.1.4 Loss in Wireless vs Varying Conditions

To have a clearer understanding of the mechanism behind the loss in Section 5.1.3, it behoved the author to run further experiments with certain scenarios in mind. The scenarios (Table 7), twelve in total, differed in that, for the first set of six scenarios, the TSN cycle time was kept constant while the send interval for non-TSN frames was varied. In the second set, the non-TSN transmission rate was kept constant, while the TSN send rate was varied. The same setups shown in Fig 24, Fig 25, and Fig 26 were used for these scenarios, but with a single workstation as non-TSN talker in those setups that had workstations (Fig 24 and Fig 26).

Table 7: Scenarios

SCENARIOS							
			period, µs	size, bytes	Expected traffic		Comment on expected traffic
CONSTANT TSN TRAFFIC							
	0	non-TSN	10	1400	1068.1	Mbps	*Network saturation*
		TSN	400	354	6.8	Mbps	
	1	non-TSN	20	1400	534.1	Mbps	
		TSN	400	354	6.8	Mbps	
	2	non-TSN	100	1400	106.8	Mbps	*More non-TSN than TSN*
		TSN	400	354	6.8	Mbps	
	3	non-TSN	400	1400	26.7	Mbps	
		TSN	400	354	6.8	Mbps	
	4	non-TSN	1600	1400	6.7	Mbps	*Same non-TSN as TSN*
		TSN	400	354	6.8	Mbps	
	5	non-TSN	5000	1400	2.1	Mbps	*Less non-TSN than TSN*
		TSN	400	354	6.8	Mbps	
CONSTANT NON-TSN TRAFFIC	6	non-TSN	400	1400	26.7	Mbps	
		TSN	1000	354	2.7	Mbps	
							More non-TSN than TSN
	3	non-TSN	400	1400	26.7	Mbps	
		TSN	400	354	6.8	Mbps	

	7	non-TSN	400	1400	26.7	Mbps	
		TSN	200	354	13.5	Mbps	
	8	non-TSN	400	1400	26.7	Mbps	*Same non-TSN as TSN*
		TSN	100	354	27.0	Mbps	
	9	non-TSN	400	1400	26.7	Mbps	*Less non-TSN than TSN*
		TSN	50	354	54.0	Mbps	
	10	non-TSN	400	1400	26.7	Mbps	
		TSN	20	354	135.0	Mbps	

Another point of difference was that the packet size was decreased for non-TSN frames from 1500 bytes to 1400 bytes. This was done because it had been noted that for the wireless leg of the transmission, when using the 1500-byte sized frames, after adding the necessary headers and bits to it during encapsulation during frame transmission, the frame became large enough that the wireless devices would break it into two fragments, sending each out individually. And with each one received, the listening wireless host would send back a wireless-ACK, signalling an acknowledgement. Only then would the receiver device note a successful reception of that frame. Changing the frame size to 1400 bytes allowed a frame to be transmitted in a single transmission; it eliminated the need for fragmentation.

The tests were iterated for three pairs of wireless and Ethernet bandwidth speeds, that is, three cases: at 11 Mbps (Eth = 10 Mbps); at 54 Mbps (Eth = 10 Mbps); at 693 Mbps (Eth = 1 Gbps); to investigate how this would affect the throughput as the different frames traversed the wireless medium. For each case, each setup was run a minimum of three times to have a mean output for the experiment. The standard deviation was also noted. This translated to at least nine simulations per scenario for each speed pair, giving a total of 27 tests per each scenario seen in Table 7. The raw data in Table 17 (linked to Table 15 and Table 16, all in Appendix C) revealed the following results presented in Fig 31.

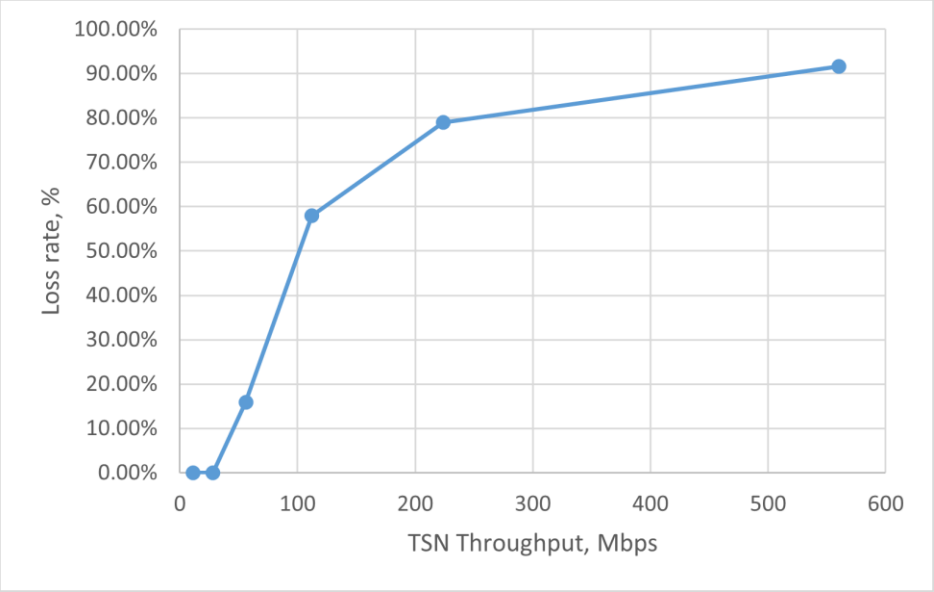

Fig 31: Loss Rate vs Throughput for TSN-only setup

The graph in Fig 31 shows the loss rates for specifically the TSN-only frames setup, in the last six scenarios, for the case of 693 Mbps wireless/1 Gbps wired. The y-axis shows the number of frames lost relative to the number of frames sent out, as a percentage; the x-axis denotes the amount of bandwidth required by the number of frames sent out. The results show that the number of TSN frames lost increased with increasing needed throughput. The loss was only kept negligible for extremely low transmission rates; below approximately 56 Mbps.

The graph in Fig 32 (produced from the results in Table 18 – Appendix C), shows the percentage of all kinds of frames lost in the first six scenarios, for the case with 693 Mbps wireless/1 Gbps wired.

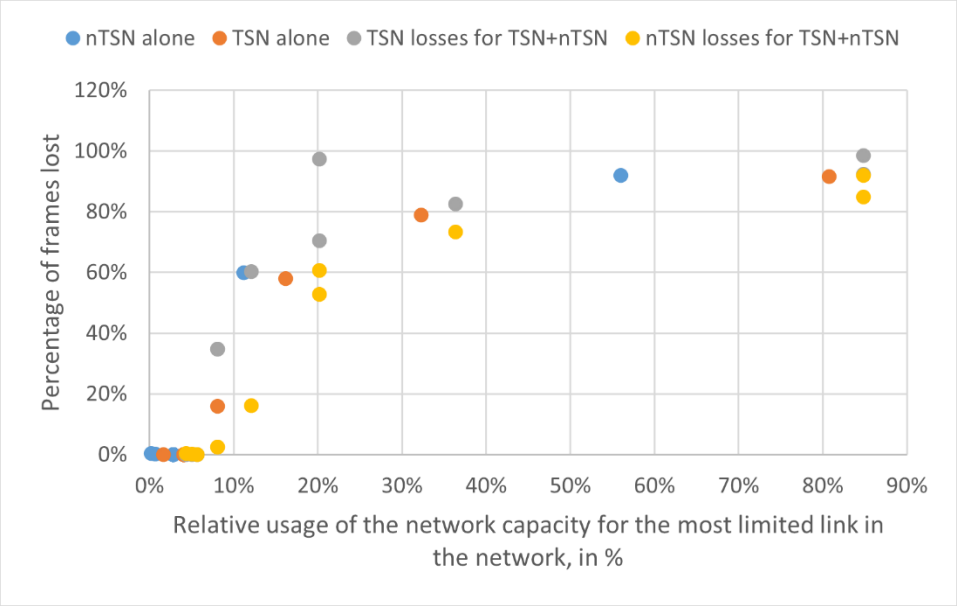

Fig 32: Frames Lost vs Relative Network Usage

The y-axis shows the percentage of frames lost; the x-axis denotes the usage of the network for the most limited link. The writer's observations were in agreement with those made from the graph in Fig 31, that the network was only reliable when its usage was extremely low. Even at relatively low network usage, the number of frames lost was still high. And it remained this way with increasing usage of bandwidth resources, as measured from the weakest link in that network. Taking the simulations in scenario 5, for example, through calculations with the given parameters, it was expected that 2.1 Mbps of best-effort traffic and 6.8 Mbps of critical traffic would be received. For this total of 8.9 Mbps, even the lowest link speeds of 11 Mbps wireless/10 Mbps wired should be sufficient. But as it happened, in the combined setup, only 1.68 Mbps and 1.9 Mbps of the respective traffics went through, totalling 3.58 Mbps. It only rose marginally (2.28 Mbps and 6.75 Mbps, totalling 9.03 Mbps) when the link speeds were drastically increased to within the gigabits-per-second range (693 Mbps wireless/1 Gbps wired). This correlates to approximately a three times increase in traffic for a 50 times increase in capacity.

5.2 Wired TSN Simulation

In the simulation results shown in 5.1, the wireless access point was reporting high loss at high rate of frames sent, and low loss only at very low required throughputs, most likely because it did not have the shaping capabilities needed to efficiently deliver TSN frames, especially in situations of high throughput. But shaping was expected to be done beforehand in the switches, so there had to be another reason. Would the switches alone perform similarly in a fully wired setup? How would varying parameters affect their output across a range of scenarios as those witnessed in the wireless setups? To answer these questions, it was decided to round up some further test scenarios, in Table 8, that involve the fully-wired setup in this

stage of the analysis. This was done to investigate just how differently the traffic was shaped in the original wired setup as opposed to the wireless one.

Table 8: Scenarios (wired)

SCENARIOS			
	Packet size and send interval		
0		period, µs	size, bytes
	non-TSN	12	1500
	TSN	400	354
	From this scenario onwards, same packet size and period		
		period, µs	size, bytes
1	non-TSN & TSN	1000	1400
2	non-TSN & TSN	500	1400
3	non-TSN & TSN	300	1400
4	non-TSN & TSN	200	1400
5	non-TSN & TSN	100	1400
6	non-TSN & TSN	50	1400
7	non-TSN & TSN	30	1400
8	non-TSN & TSN	20	1400

The simulation involving the fully-wired combined setup in Fig 28 reported favourable results, but this test involved different parameters for both the critical and best-effort portions of the traffic. For example, TSN traffic packets were much shorter, which gave them an advantage and hindered a direct comparison. It was decided that, in order to analyse the extent of the bias towards each kind of traffic, the parameters be kept the same for both traffics. And so similar tests were repeated for this setup, using a single workstation and frame size kept at 1400 bytes for both frame types. The same send interval was also used, and it was decreased in stages for a total of eight iterations. These eight scenarios each had three cases for when Ethernet speed was at: 10 Mbps, 100 Mbps and 1 Gbps. Each test was repeated for a minimum of three runs in order to record the mean and standard deviation in case of any resulting errors or unusual discrepancies. The packets lost were also recorded. The parameters and raw results for the different scenarios tested are as shown in Table 8, Table 19 (Appendix C), and the loss results presented here in Table 9.

Table 9: Relative Loss Magnitude (wired)

COUNTING LOST FRAMES		Period of sending fra	
		nonTSN, us	TSN, us
	Scenario 1	1000	
	Scenario 2	500	
	Scenario 3	300	
	Scenario 4	200	
	Scenario 5	100	
	Scenario 6	50	
	Scenario 7	30	
	Scenario 8	20	

These results can be summarised in the graph in Fig 33.

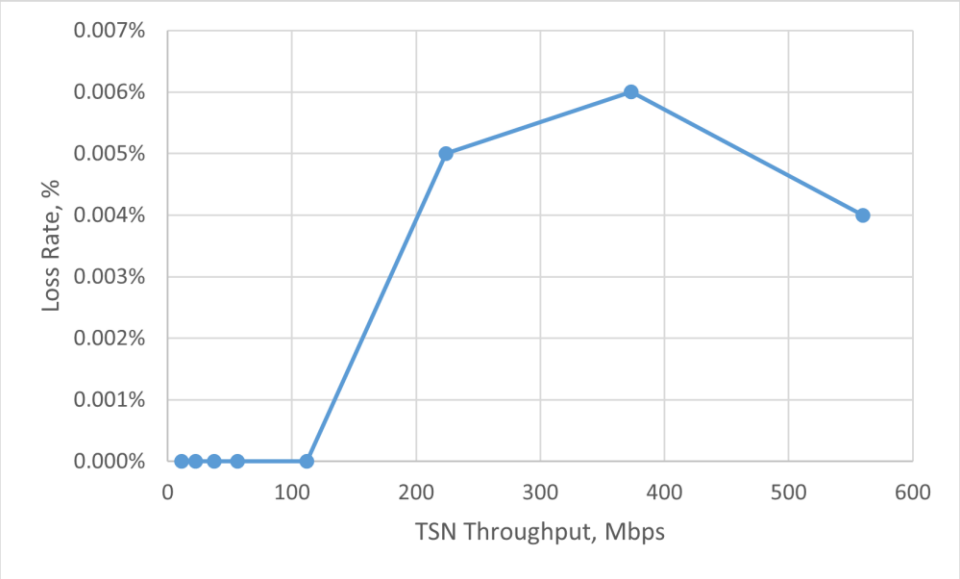

Fig 33: Loss Rate vs Throughput (wired) for TSN-only frames setup at 1 Gbps bandwidth

The graph shows the loss rates for specifically the TSN-only frames setup, in the eight scenarios conducted, solely for the case of the 1 Gbps bandwidth. The y-axis again shows the number of frames lost relative to the number of frames sent out, as a percentage; the x-axis denotes the amount of bandwidth required according to the number of frames sent out. In comparison to Fig 31, the loss data for the raw results for Fig 33 had to be adjusted to three decimal places because the losses were just that low; nothing exceeding 0.01 %.

5.3 Wireless Simulation with Traffic Split at Access Points

Based on the above results, it was evident that all losses happened at the point the medium switched from wired to wireless. Considering that there is no existing code for wireless TSN

available in NeSTiNg and the author wanted to attempt a practical solution, they recognised the need to induce a means of separating traffic for the wireless channel. The access points had no inherent traffic-prioritising mechanism that was as efficient as TSN priority classes in the TSN-enabled switches. To remedy this, two access points were used at one end to split the traffic from the *switchB* interface (Fig 34). This was done so that TSN traffic could be assigned to one dedicated access point, while non-TSN traffic was forwarded to another, with aims of apportioning specific routes to the respective flows so that each access point only handled one type of traffic.

Fig 34: 2 APs, on one end

Then this setup was tested against the wired one, whose results were revealed in Fig 33. Similar to the wired case, using the same frame size and intervals for both kinds of frames (TSN and non-TSN), allowing for a side-by-side comparison, the frame send intervals were varied uniformly across both setups, for Ethernet at 100 Mbps and 1 Gbps. The frame size was kept constant (1400 bytes). 693.3 Mbps Wi-Fi was used for the wireless implementation.

The parameters and raw results for the different scenarios tested are as shown in Table 20 and Table 21 (Appendix C), and the loss results are shown in Table 10 below.

Table 10: Relative Loss Magnitude (wireless: 2 APs, on one end)

COUNTING LOST FRAMES		Period of sendi
		nonTSN, us TSI
	Scenario 1	1000
	Scenario 2	500
	Scenario 3	300
	Scenario 4	200
	Scenario 5	100
	Scenario 6	50
	Scenario 7	30
	Scenario 8	20

This implementation's results can be summarised in Fig 35.

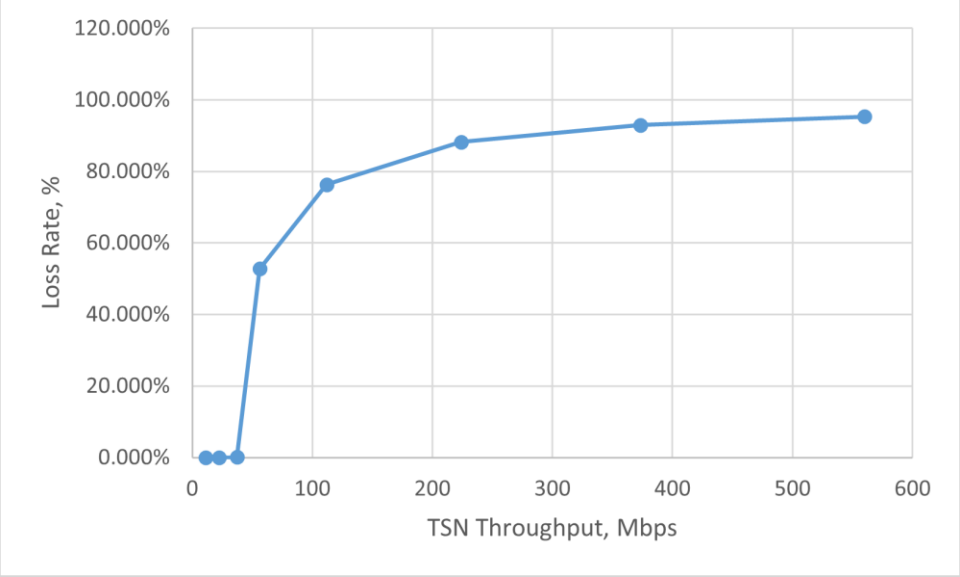

Fig 35: Loss Rate vs Throughput (wireless: 2 APs, on one end)

In the graph, the loss rates for the TSN-only frames setup for the eight scenarios conducted, are documented, solely for the case of the 1 Gbps bandwidth. The y-axis again shows the number of frames lost relative to the number of frames sent out, as a percentage; the x-axis denotes the amount of bandwidth required according to the number of frames sent out. The loss rates were kept around 0 % when the channel usage was up to 37 Mbps, and more importantly, the frame send interval was equivalent to 300 µs. When the interval was decreased to 200 µs (and so throughput increased to 56 Mbps), the loss rate shot up by 50 %; a notable increase indeed. This suggested that the access point was overwhelmed between these two points, 300 µs and 200 µs.

5.4 Results against TSN Requirements

The results show that though the model's reliability suffered due to its low achievable throughput numbers, it was not off the mark in all sectors. The writer endeavoured to satisfy the main requirements needed for a TSN communication, as listed in Section 3.3.1.

5.4.1 Requirement I – Reliable Channel

For requirement I; the wireless model could not guarantee a reliable channel as significant losses occurred for TSN frames. The loss numbers only reduced when channel usage was kept quite low. This has been discussed in detail in the prior sections, Section 5.1.4 and Section 5.3. In summary, the experiments ran showed that, for whichever case that was considered, TSN traffic only, non-TSN traffic only, or mixed traffic, the most frames were delivered successfully when the usage of the channel was under around 10 %, as measured from the most limited link. Fig 32 shows losses were kept below 3 % in such cases. In this regard, the channel and model could not be classified as reliable.

5.4.2 Requirement II – Low Latency

For requirement II; although all the experiments had a time limit of one second, where even the setup with the most constrained parameters was able to send out frames within the hundreds, because the reliability element was missing, the latency could not be properly analysed. Reliability took precedence over latency, as the delimitations in Section 1.7 specify.

5.4.3 Requirement III – Capacity

For the third requirement; the broad range of experiments performed showed that the setup was able to operate to cater for a range of speeds. The initial set of simulations run were for: 11 Mbps/10 Mbps, 36 Mbps/10 Mbps, 36 Mbps/100 Mbps, 54 Mbps/10 Mbps, 54 Mbps/100 Mbps, 54 Mbps/1000 Mbps, 693 Mbps/ 100 Mbps, 693 Mbps/1 Gbps, for wireless/Ethernet speeds respectively. Since most industrial applications are not bandwidth intensive, the findings reported that a steady capacity of 100 Mbps is adequate (Section 3.3.1). The model could support this. It was noticed that when run at higher bandwidth speeds it could offload the constrained links because the channel utilisation was lower.

5.5 Final Thoughts

The last main objective in this work, as highlighted in Section 1.5, was to use the model made to study and highlight the features that a wireless TSN network needs in order to be reusable in environments requiring strict timing, like mission-critical Internet of Things (IoT) applications.

The wireless access point was the single point for network congestion in the experiments, from which bred the majority of losses in the network (as network congestion causes packet losses and increased latency periods). It can now be reasoned that, because the access point is unaware of the time-sensitive protocols implemented before it, this is what happens. All

the results lead to this. The following discussion is meant to highlight the consequences of this.

QoS parameters can be defined for an access point to determine how it deals with TCP/UDP traffic. Using the Enhanced Distributed Channel Access (EDCA) mechanism, which is a part of the MAC's HCF (Hybrid Coordination Function) submodule, the MAC layer applies QoS for packets belonging to one of four access categories, namely, voice, video, best effort and background, in order of decreasing priority. But in the case of the novel model in [23], no higher layer protocols are used; it is purely Ethernet frames. So, after applying QoS to the wireless stations, the frames were disguised as voice or video services, for example, in order to change how the access point dealt with the frames, TSN or otherwise.

The writer gave the highest priority to the TSN frames. Even after doing this, the results were underwhelming, and in some cases, even worse than before this QoS was applied. In the course of the experiments, it was verified that the QoS function did not work in unison with TSN. The situation thus far rendered the access point, for all intents and purposes, TSN-unaware. Since it did not have the scheduling mechanisms needed, it treated all frames equally. Operating in a medium such as Wi-Fi, where there is already a high chance of collisions, the situation was worsened since access points do not forward frames as they are received; they have to first check and acknowledge the availability of the channel. The access points were not as responsive to the incoming message flow; the queues quickly got full; critical frames were dropped in the process.

To briefly outline the main points alluded to in Section 2.3 and Section 4.3.1.1, on the time-sensitive aspects of TSN, TSN packets are tagged with a special VLAN ID, among other things, which identifies the way they will be handled by switches (in terms of priority and the urgency of the traffic flow). And just as important, queues are defined in the switches that helps to directly shape the processing of the network traffic they receive. The earlier-mentioned attributes (VLAN ID and others) in the frames tell the switches how to process frames and in which queues. With the access point, even when a low back-off interval was assigned so that frames waited for shorter durations in queues, this was applied uniformly for both critical and non-critical frames. Its innate abilities do not allow it to distinguish between TSN vs non-TSN frames, and so they were treated as the same kind of traffic.

Secondly, even with an optimum combination of parameters (such as in Table 5 of Section 4.6), the access point still could not process frames fast enough, which likely filled up the queues in the wireless devices and led to many frames being dropped. Conclusively, the results point to the fact that current wireless standards need more work for the case of non-wireline TSN. One possible solution would be for access categories for the wireless medium to redefine their priority scheme to include support for a TSN traffic class. Further, another could be to borrow from the DetNet specification and have a bounded or maximum bandwidth existing

for different kinds of traffic in the network. That way, the best-effort traffic will not be allowed to overwhelm the network and choke out the critical traffic as its throughput will be capped at a certain limit. Expounding on this, say, no more best-effort traffic is allowed to flow through the network past a certain point, maybe unless the critical traffic throughput is kept above a certain level or above a certain percentage of the whole network's traffic at any time.

As per Section 2.3.3, Seijo et al. [58] similarly propose amendments to the 802.11 standard in accordance with their observations in their hybrid TSN solution, consisting of wired and wireless portions. They similarly found that the maximum amount of traffic had to be lowered to allow their implementation to function optimally, among other downsides like manual configuration of the maximum transmission unit on each access point, so that Wi-Fi nodes send out small frames. Another among their reasoning was to stop non-critical traffic in Wi-Fi nodes from transmitting during periods reserved for real-time traffic.

Per Section 2.3.3., Genc and Del Carpio [59], also recognising the amendments needed, and they go on to actually modify the Wi-Fi MAC layer in their simulated TSN solution. They add a new access category for higher priority traffic. In their simulation, they proved that this new QoS access category significantly improved performance in jitter and delay. Similarly, as in this work, [59] could not guarantee reliability as required by TSN applications. Their results show that the majority of packets benefitted from this, while the rest were not delivered timely. In order to prevent the jitter and delay in the network from worsening, the packets had to be dropped. Consequently, this contributes to higher packet losses, and goes against the goals of reliability. As mentioned in Section 2.3.3, [59] conclusively state that their model is suitable for those use cases in industry which do not need very high reliability with reference to latency, delay and packet loss.

Once 5G-TSN standardisation is at a later stage and more fully developed, the concept of integrating the 5G system into TSN as a logical or transparent bridge will hopefully yield higher performance results than the counterpart integration of Wi-Fi as detailed in this work. This is mainly because, as discussed in the simulation model presented in [65], in Section 2.3.4, 5G functions in frequencies that are licensed by authorities, and so many issues of interference will be avoided. According to [65], since 5G development is currently at a state where the availability of devices supporting full Release 16 is not yet granted, simulative approaches are most suitable for large scale networks due to the reduced time for deployment and their flexibility.

The findings from this section are also consistent with those presented in the author's paper [86] that was published in the 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS 2021), which provides an evaluation of broadband technologies, including Wi-Fi, from an industrial time sensitive networking perspective.

5.6 Chapter Summary

This chapter's offerings include a comparison of the throughput performance of the proposed wireless TSN implementation against a fully-wired conventional setup through several tests. The wireless model was seen to under-achieve, and the analysis pointed towards the internal mechanisms in the wireless access point being TSN-unaware, and so not distinguishing between time-triggered vs non-time-triggered traffic. The only scenarios where the model performed adequately were in those where the capacity of the channel being used was kept very low, below 10 % of the channel's capacity.

The author used the model made to answer their initial objectives. The last major objective was to use the model to study and highlight the features that wireless TSN would need to be reusable in environments with stringent timing requirements. The model was especially helpful in helping to analyse these essential features. What it revealed was that, although it can be visualised, the state-of-the-art tools are not as yet ready to conceive an efficient wireless TSN implementation. At least, not without the needed redesigning to extend the current wireless device protocols to coordinate the timely functions even over the wireless interfaces and radio. It requires modifying some parts of the wireless components, such as the radio of access points, to allow it to distinguish between and separate out the more urgent frames from the general traffic. Moreover, wireless standards require further development to accommodate TSN traffic; possibly an added dimension in their access categories to allow this traffic differentiation.

The graphs presented herein (Chapter 5) and the full tables and raw results from Appendix C can be viewed from the examples folder, under the main simulations folder in the source code provided. More results can be generated as is desired to view different aspects of the models designed.

6. Conclusion

6.1 Overall Findings

TSN is one of the preferred methods for deploying critical communications in industry today. The state-of-the-art provides a way for multiple services to run alongside TSN through the same Ethernet architecture. As a collection of Ethernet standards, it enhances the efficiency of standard Ethernet, making it more reliable for the deterministic use case, of which reliability is a staple. The systems forming part of the TSN infrastructure consists of TSN-aware bridges or switches which forward frames and relay messages within their scheduled time windows. Through principles of synchronisation, scheduling and frame pre-emption, these components can shape the traffic on the factory floor. Work towards the wireless realm is ongoing. As factory floors become more automated, the prospect of wireless TSN becomes a noteworthy research direction. The factory floor of the future is flexible to allow for reconfigurability of the plant's layout. The trend sees present machinery moving towards wireless alternatives to make use of the increased mobility that industrial robots can offer. Reliability is a very present concern in critical IoT applications. If they are to be deployed wirelessly, they need to assure the level of time-sensitivity that the current wired TSN displays. But as far as conventional means go, this TSN and any other deterministic mission-critical communications in industry are only ever conducted through physical means.

This research work was primarily directed towards producing a simulation model that can approach TSN through means of Wi-Fi, in order to be used in scenarios with stringent timing requirements like present TSN implementations do over Ethernet. The proposed simulation model was produced by applying techniques proffered by current TSN research directions of modifying the available state-of-the-art components and software. The methodology thus considered the OSI model in order to define the changes that needed to be made. This work was developed using the NeSTiNg framework, running atop INET and the OMNeT++ simulator platform. This allowed for discrete event simulations of several models that the writer developed to aid in their work.

This work has successfully observed wireless TSN working in conditions where the usage of the channel was under around 10 %, as measured from the most limited link. Although, when proceeding past operational ranges, the wireless models produced were not readily able to support TSN; the work entailed further modification of certain elements such as interfaces to allow for reception and processing of frames tagged with TSN modifiers. After establishing that a TSN frame could be transmitted from end to end, the writer was able to use several experiments to determine how the prototype performed in networks with mixed traffic types.

As pertains to the first two research questions, the author discussed how industrial communications in wired and wireless channels differ. Guided by a thorough review of the literature, the author determined the answer to be in those components of wired channels

that make them preferable for deterministic communications. The wired TSN switches can differentiate between the different kinds of traffic they carry. This allows for scheduling mechanisms to shape the network efficiently. With the introduction of wireless access points to carry the traffic wirelessly, the critical frames cannot be distinguished from non-critical ones. The situation could not be alleviated even through the use of internal QoS mechanisms. This lack of bias was one of the main factors that affected the proposed wireless TSN and caused it to perform sub-standardly.

The third research question called the work to address how to transmit TSN and non-TSN frames in a network with these two traffics combined. The cause of concern here was that since the traffic would be propagating in a wireless medium, that is Wi-Fi, the network would not be as reliable. The writer attempted to contend with this in the parameters by increasing the restoration rate of the wireless devices, such as the collision window and back-off counter settings, to aid them in speedier "recoveries" after they experienced collisions. The network parameters were modelled after the wired TSN setup to reduce the delay between interconnected components. Lastly, in beginning to answer the fourth research question of whether the state-of-the-art components in the standard TSN implementation can be translated or replaced to allow it to cater for the wireless design, this was addressed via the experiments. Also discussed are the differences in performances in relation to the throughput and frames lost in communication, through the results collected. The results show that the quality of the network was thoroughly degraded by the TSN-unaware switch, which could not differentiate between the different traffic, and, even with an optimum combination of parameters, it still could not process frames fast enough.

6.2 Significance of the Study

A summary of the main contributions of this work follow:

- The simulation model tries to run in an isolated wireless environment, which is an advancement on prior TSN approaches in practical and simulation environments.
- Further, it showed how network elements can be fused to achieve a more flexible design for applications on the plant-floor. This was achieved by combining wireless hosts and devices with cabled infrastructure and switches. The proposed model demonstrated how these component devices can be integrated and reused for TSN communication. The factory floor of the future should be versatile to allow for easy repositioning of machinery; wireless devices suit this layout.
- The model produced can potentially be used to highlight the features standing in the way of wireless TSN being deployed in ULL networks.
- In that same line of thought, practical implications of the study include aiding in evaluating the performance of TSN. The work could likely also aid novice users in

knowing where to begin in setting up TSN LAN to run and manage their own services. For example, testing out the efficiency of a planned factory-floor network implementation before installing it.

6.3 Limitations, Recommendations and Future Work

The INET framework allowed different functions to be programmed and integrated into the model to achieve the desired modus operandi, by modifying or extending the module's source code. Although it supports the forwarding of TSN traffic, the proposed model does not have the features necessary to allow the wireless radios to do so in a timeous manner. It only successfully forwards these when the traffic passed through is kept at very low thresholds. In this way it can certainly be improved.

In earlier sections of this work (Section 4) it was identified that the synchronisation feature for NeSTiNg TSN to model the IEEE Standard 802.1AS was not completely developed. In [23], the authors mention their plans to integrate further TSN mechanisms in the future, such as the asynchronous traffic shaper and clock synchronisation protocols. On reaching out to them about this, more specifically on the matter of global clock synchronisation (as opposed to local clock sync, which is what is used in the NeSTiNg tool), the writer was able to confirm that this part of the future work is indeed very much open. As a possible basis for the future work of a global clock for synchronisation, this work does include in (Appendix B) an algorithm that breaks down how the BMCA and 1588 PTP sync mechanism that is needed for syncing devices could be designed into the framework, detailing the operation flow.

Further work also extends to using the wireless model developed and introducing to it external disturbances that negatively affect the performance of the network to mimic real-life scenarios. From the review conducted, the prime source of such would be the degradation caused by signals that induce interference. By altering settings in the model, interference can be modelled from different sources, and their signal strengths can be varied to simulate fading path effects. It would provide an insight into the probability of successful communication based on altering a mix of other variables: by varying the strength of different signals over a set time and gauging how this affects the communication across different distances (communication ranges), interference ranges, transmitter radio powers and other variables.

Another among the desired functionalities that being present would render for a more time-sensitive communication, regardless of the mobile technology used, Wi-Fi, 5G or other, would have to be the capability for redundancy. The IEEE TSN Standard **802.1CB** – FRER. This would only help to succour the standard of reliability by reducing packet loss since it replicates streams to carry the same frames of information. With one failing, there would be others that pick up the transmission attempt, ensuring that the message still gets to the target in the planned time window. Speaking on Wi-Fi, version 802.11ac supports up to 8 streams, and the

writer proposes that these could work in a similar manner as the redundant streams for 802.1CB – a possible future research direction to extend the functionality of the tool, and ultimately test out more scenarios of wireless TSN before deployment in a production environment.

The advent of Wi-Fi 6 (802.11ax) as the new widely adopted wireless LAN standard is upon us. It is the view of certain authors such as those in [34] that Wi-Fi, even in version 7 (802.11be), will not be able to guarantee a fully deterministic communication because of having to share spectrum in license-exempt frequencies. Of course, not all communications are urgent, so even though the wireless access point has to contend with many other devices external to it sharing the spectrum, it can greatly mitigate the many delays that originate internally that affect its operations' determinism by selecting these more urgent communications over the best-effort traffic when it detects or receives them.

The main problem that comes with this is that it becomes a requisite that the access point be able to pre-empt non-critical communications in favour of the more critical ones sent out by applications with stringent timing requirements. It should further have some mechanism that allows for the buffering of pre-empted frames as they wait to be transmitted. It is the opinion of the writer and others in [24], [33], [34] that the ideal solution to this would entail redesigning the wireless system to be capable of managing both time-sensitive and non-time-sensitive traffic. Introducing these new features could also come in the form of developing new hardware and technology. As noted by the cohort comprised of the IEEE Wireless Networking Group Standing Committee [57], it is not expected that such a design which purposely introduces into their equipment the ability to voluntarily stop transmitting or receiving some communications to favour others would be easily or quickly adopted by the manufacturers of Wi-Fi devices. But its potential to maximise the efficiency in critical wireless communications is something that the writer believes warrants exploration.

Among the recommendations, the current standards for wireless access categories may need to be redefined to include support for TSN traffic, which it does not currently do. This is one way to go. Another possible solution would be to define an upper bound on capacity for the network traffic, such that the best-effort traffic class cannot exceed the critical traffic past a certain amount or proportion. In this way, the urgent traffic cannot get choked out as it did in this work's experiments.

The solution proposed in this work tries to leverage the current wireless design in all its potential and use the tools presently at its disposal to integrate it with TSN. The results indeed suggest a move towards a reimagining of the design that renders the Wi-Fi components more flexible and gives them a fuller control over the TSN domain with the capability to carry and support all other communications alongside this domain. Further research in the direction of

conceptualising this design of the physical architecture of these wireless devices could help ascertain or confirm this premise.

References

[1] L. Columbus, "2018 Roundup Of Internet Of Things Forecasts And Market Estimates," *Forbes*, Dec-2018. [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2018/12/13/2018-roundup-of-internet-of-things-forecasts-and-market-estimates/#386fb5e37d83. [Accessed: 25-Nov-2019].

[2] Daimler AG, "Mercedes-Benz Factory 56: Voll flexible Produktion," 2020. [Online]. Available: https://www.youtube.com/watch?v=-iJu8EU3_YU. [Accessed: 09-Mar-2021].

[3] A. Nasrallah *et al.*, "Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 1, pp. 88–145, 2019, doi: 10.1109/COMST.2018.2869350.

[4] S. B. H. Said, Q. H. Truong, and M. Boc, "SDN-based configuration solution for IEEE 802.1 time sensitive networking (TSN)," *ACM SIGBED Rev.*, vol. 16, no. 1, pp. 27–32, Feb. 2019, doi: 10.1145/3314206.3314210.

[5] K. B. Stanton, "Distributing Deterministic, Accurate Time for Tightly Coordinated Network and Software Applications: IEEE 802.1AS, the TSN profile of PTP," *IEEE Commun. Stand. Mag.*, vol. 2, no. 2, pp. 34–40, Jun. 2018, doi: 10.1109/MCOMSTD.2018.1700086.

[6] L. Lo Bello and W. Steiner, "A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems," *Proc. IEEE*, vol. 107, no. 6, pp. 1094–1120, Jun. 2019, doi: 10.1109/JPROC.2019.2905334.

[7] IEEE Standards Association, *802.1AS-2011 - IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*, no. March 30. IEEE, 2011.

[8] IEEE Standards Association, *P802.1AS-Rev/D8.1, Aug 2019 - IEEE Draft Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications*. IEEE, 2019.

[9] IEEE Standards Association, *IEEE Standard for Local and metropolitan area networks -- Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic*. 2016.

[10] IEEE Standards Association, *IEEE Standard for Local and metropolitan area networks -- Bridges and Bridged Networks -- Amendment 26: Frame Preemption*. 2016.

[11] M. D. Johas Teener and G. M. Garner, "Overview and timing performance of IEEE 802.1AS," in *2008 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2008, pp. 49–53, doi: 10.1109/ISPCS.2008.4659212.

[12] G. A. Ditzel III and P. Didier, "Time Sensitive Network (TSN) Protocols and use in EtherNet/IP Systems," in *ODVA 2015 ODVA Industry Conference & 17th Annual*

Meeting, 2015.
[13] National Instruments, "What's the Difference Between TSN (802.1AS) and IEEE-1588?," *Knowledgebase Article*, Sep-2018. [Online]. Available: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019V0iSAE&l=en-ZA. [Accessed: 25-Nov-2019].
[14] H. Cho, J. Jung, B. Cho, Y. Jin, S.-W. Lee, and Y. Baek, "Precision Time Synchronization Using IEEE 1588 for Wireless Sensor Networks," in *2009 International Conference on Computational Science and Engineering*, 2009, vol. 2, pp. 579–586, doi: 10.1109/CSE.2009.264.
[15] D. A. Klimov, "Features of Synchronization in Wireless Communication Networks," in *2018 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*, 2018, pp. 1–3, doi: 10.1109/SYNCHROINFO.2018.8457055.
[16] Oscilloquartz, "Synchronizing LTE-advanced and 5G radio access networks - Emerging mobile services and IoT demand precise timing," in *ADVA Optical Networking*, 2018, pp. 1–8.
[17] TTTech, "IEEE TSN (Time-Sensitive Networking): A Deterministic Ethernet Standard," *TTTech*, 2015.
[18] S. Brooks and E. Uludag, "Time-Sensitive Networking: From Theory to Implementation in Industrial Automation," *TTTech Intel White Pap.*, Oct. 2018.
[19] IEEE Standards Association, *IEEE Standard for Local and metropolitan area networks--Frame Replication and Elimination for Reliability*. 2017.
[20] IEEE Standards Association, *IEEE Standard for Local and Metropolitan Area Networks--Bridges and Bridged Networks -- Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements*, vol. 2018. 2018.
[21] TTTech Computertechnik AG, "Time Sensitive Networking (TSN) - TTTech," 2018. [Online]. Available: https://www.tttech.com/technologies/time-sensitive-networking-tsn/. [Accessed: 06-Feb-2020].
[22] J. Jiang, Y. Li, S. H. Hong, A. Xu, and K. Wang, "A Time-sensitive Networking (TSN) Simulation Model Based on OMNET++," in *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2018, pp. 643–648, doi: 10.1109/ICMA.2018.8484302.
[23] J. Falk *et al.*, "NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++," in *2019 International Conference on Networked Systems (NetSys)*, 2019, pp. 1–8, doi: 10.1109/NetSys.2019.8854500.
[24] S. F. Bush and G. Mantelet, "Industrial Wireless Time-Sensitive Networking: RFC on the Path Forward," in *Avnu Alliance® White Paper*, 2018, pp. 1–12.
[25] E. Khorov, "Wi-Fi Time Sensitive Networking," *IEEE Mentor*, Nov-2017. [Online]. Available: https://mentor.ieee.org/802.11/dcn/17/11-17-1734-00-0wng-wtsn.pptx. [Accessed: 30-Sep-2019].

[26] X. Yang, D. Scholz, and M. Helm, "Deterministic Networking (DetNet) vs Time Sensitive Networking (TSN)," in *Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM)*, 2019, pp. 79–84, doi: 10.2313/NET-2019-10-1_15.

[27] Z. Shen, "Reliable wireless communication for factory automation," *Ind. Ethernet B.*, no. 84/14, 2014.

[28] K. Zwerina, "Standards-based real time Ethernet now off-the-shelf," *Ind. Ethernet B.*, no. 29/31, 2005.

[29] Department of Science & Technology, "Innovation Towards a Knowledge-based Economy: Ten-Year Plan for South Africa," 2008.

[30] C. H. Kai and S. C. Liew, "Temporal Starvation in CSMA Wireless Networks," in *2011 IEEE International Conference on Communications (ICC)*, 2011, pp. 1–6, doi: 10.1109/icc.2011.5963320.

[31] C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. M. Vilches, "Time-Sensitive Networking for robotics," Apr. 2018.

[32] A. Mildner, "Time Sensitive Networking for Wireless Networks-A State of the Art Analysis," *Semin. IITM WS 18/19*, 2019, doi: 10.2313/NET-2019-06-1_07.

[33] D. Bankov, E. Khorov, A. Lyakhov, and M. Sandal, "Enabling real-time applications in Wi-Fi networks," *Int. J. Distrib. Sens. Networks*, vol. 15, no. 5, p. 155014771984531, May 2019, doi: 10.1177/1550147719845312.

[34] T. Adame, M. Carrascosa, and B. Bellalta, "Time-Sensitive Networking in IEEE 802.11be: On the Way to Low-latency WiFi 7," *Dep. Inf. Commun. Technol. Univ. Pompeu Fabra*, Dec. 2019.

[35] A. Weder, "Time Sensitive Networking: An Introduction to TSN," *Fraunhofer IPMS TSN White Pap.*, 2019.

[36] PROFINET, "PROFINET - The Solution Platform for Process Automation," *PI (PROFINET Int. White Pap.*, Jun. 2015.

[37] M. Rostan, "Industrial Ethernet Technologies: Overview and Comparison," in *ETG (EtherCAT Technology Group) Industrial Ethernet Seminar Series*, 2014.

[38] L. Wang, M. Li, J. Qi, and Q. Zhang, "Design Approach Based on EtherCAT Protocol for a Networked Motion Control System," *Int. J. Distrib. Sens. Networks*, vol. 10, no. 2, p. 750601, Feb. 2014, doi: 10.1155/2014/750601.

[39] S. Nsaibi, L. Leurs, and H. D. Schotten, "Formal and simulation-based timing analysis of industrial-ethernet sercos III over TSN," in *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2017*, 2017, vol. 2017-Janua, pp. 1–8, doi: 10.1109/DISTRA.2017.8167670.

[40] Wikipedia contributors, "SERCOS III," *Wikipedia, The Free Encyclopedia*, Sep-2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=SERCOS_III&oldid=918406786. [Accessed: 25-Nov-2019].

[41] S. Szancer, P. Meyer, and F. Korf, "Migration from SERCOS III to TSN-Simulation Based Comparison of TDMA and CBS Transportation," in *Proceedings of the 5th International OMNeT++ Community Summit*, 2018, pp. 52–62, doi: 10.29007/c5td.
[42] ODVA, "Technology Overview Series: EtherNet/IP - CIP on Ethernet Technology," *ODVA Technol. Overv. Ser.*, Mar. 2016.
[43] AMMC, "6 Major Industrial Ethernet Technologies," *AMMC (Applied Machine & Motion Control) Golden Motion*, Jul-2018. [Online]. Available: https://ammc.com/6-major-industrial-ethernet-technologies/. [Accessed: 25-Nov-2019].
[44] S. Zuponcic, "EtherNet/IP™ for Real-Time, Machine-to-Machine Control," *ODVA White Pap.*, Oct. 2015.
[45] J. Hibbard, "5 Real-Time, Ethernet-Based Fieldbuses Compared," *AIA (Automated Imaging Assoc. Vis. Online White Pap.*, Mar. 2016.
[46] Real Time Automation Inc., "EtherNet/IP: An Application Layer for Industrial Automation - Real Time Automation, Inc.," in *Real Time Automation Inc. White Paper*, 2019, pp. 1–4.
[47] J. S. Rinaldi, "What is CIP? - Real Time Automation, Inc.," *Real Time Automation, Inc.*, Aug-2018. [Online]. Available: https://www.rtautomation.com/rtas-blog/what-is-cip/. [Accessed: 24-Feb-2020].
[48] Sercos III, "Sercos - The Automation Bus." [Online]. Available: https://www.sercos.org/. [Accessed: 18-Dec-2020].
[49] J. S. Rinaldi, "EtherCAT vs. EtherNet/IP - Real Time Automation, Inc.," *Real Time Automation, Inc.*, Oct-2019. [Online]. Available: https://www.rtautomation.com/rtas-blog/ethercat-vs-ethernet-ip/. [Accessed: 24-Feb-2020].
[50] P. Drahos, E. Kucera, O. Haffner, and I. Klimo, "Trends in industrial communication and OPC UA," in *2018 Cybernetics & Informatics (K&I)*, 2018, vol. 2018-Janua, pp. 1–5, doi: 10.1109/CYBERI.2018.8337560.
[51] P. Zimmermann, E. Axmann, B. Brandenbourger, K. Dorofeev, A. Mankowski, and P. Zanini, "Skill-based Engineering and Control on Field-Device-Level with OPC UA," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, vol. 2019-Septe, pp. 1101–1108, doi: 10.1109/ETFA.2019.8869473.
[52] OPC Foundation, "Press Releases Archives - OPC Foundation." [Online]. Available: https://opcfoundation.org/news/press-releases/. [Accessed: 01-Mar-2020].
[53] P. Lutz, "OPC UA 'Field Level Communications Initiative,'" in *OPC Day Finland, Helsinki*, 2019.
[54] OPC Foundation, "Initiative: Field Level Communications (FLC) OPC Foundation extends OPC UA including TSN down to field level," 2019.
[55] D. Cavalcanti, G. Venkatesan, and C. Cordeiro, "PAW-IETF 104-Prague Time-Sensitive applications support in 802.11ax and 802.11be (EHT)," 2019.
[56] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling Fog Computing for

Industrial Automation Through Time-Sensitive Networking (TSN)," *IEEE Commun. Stand. Mag.*, vol. 2, no. 2, pp. 55–61, Jun. 2018, doi: 10.1109/MCOMSTD.2018.1700057.

[57] L. Wang, "WNG Meeting Minutes of 2017-November Orlando Meeting," in *Wireless Next Generation (WNG) Standing Committee (SC)*, 2017, pp. 1–3.

[58] O. Seijo, Z. Fernandez, I. Val, and J. A. Lopez-Fernandez, "SHARP: Towards the Integration of Time-Sensitive Communications in Legacy LAN/WLAN," in *2018 IEEE Globecom Workshops (GC Wkshps)*, 2018, pp. 1–7, doi: 10.1109/GLOCOMW.2018.8644124.

[59] E. Genc and L. F. Del Carpio, "Wi-Fi QoS Enhancements for Downlink Operations in Industrial Automation Using TSN," in *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2019, vol. 2019-May, pp. 1–6, doi: 10.1109/WFCS.2019.8757992.

[60] C. Mannweiler *et al.*, "Reliable and Deterministic Mobile Communications for Industry 4.0: Key Challenges and Solutions for the Integration of the 3GPP 5G System with IEEE - VDE Conference Publication," in *Mobile Communication - Technologies and Applications; 24. ITG-Symposium*, 2019, pp. 1–6.

[61] S. Sultana, "LS on 3GPP 5G System support for integration with IEEE TSN networks," in *Standards Association (SA) Working Group #2 (WG2)*, 2019, pp. 1–2.

[62] D. Chandramouli, "5G for Industry 4.0," *3GPP*, 2020. [Online]. Available: https://www.3gpp.org/news-events/2122-tsn_v_lan. [Accessed: 23-Aug-2020].

[63] A. Neumann, L. Wisniewski, R. S. Ganesan, P. Rost, and J. Jasperneite, "Towards integration of Industrial Ethernet with 5G mobile networks," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2018, vol. 2018-June, pp. 1–4, doi: 10.1109/WFCS.2018.8402373.

[64] A. Larranaga, M. C. Lucas-Estan, I. Martinez, I. Val, and J. Gozalvez, "Analysis of 5G-TSN Integration to Support Industry 4.0," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2020, vol. 2020-Septe, pp. 1111–1114, doi: 10.1109/ETFA46521.2020.9212141.

[65] L. Martenvormfelde, A. Neumann, L. Wisniewski, and J. Jasperneite, "A Simulation Model for Integrating 5G into Time Sensitive Networking as a Transparent Bridge," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2020, vol. 2020-Septe, pp. 1103–1106, doi: 10.1109/ETFA46521.2020.9211877.

[66] J. K. Ray, P. Biswas, R. Bera, S. Sil, and Q. M. Alfred, "TSN enabled 5G Non Public Network for Smart Systems," in *2020 5th International Conference on Computing, Communication and Security (ICCCS)*, 2020, pp. 1–6, doi: 10.1109/ICCCS49678.2020.9277016.

[67] M. Gundall, C. Huber, P. Rost, R. Halfmann, and H. D. Schotten, "Integration of 5G with TSN as Prerequisite for a Highly Flexible Future Industrial Automation: Time

Synchronization based on IEEE 802.1AS," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, 2020, vol. 2020-Octob, pp. 3823–3830, doi: 10.1109/IECON43393.2020.9254296.

[68] IETF, "Deterministic Networking (detnet) - Documents," *Datatracker - Internet Engineering Task Force*. [Online]. Available: https://datatracker.ietf.org/wg/detnet/documents/. [Accessed: 18-Dec-2020].

[69] Cisco, "Time-Sensitive Networking: A Technical Introduction White Paper Time-Sensitive Networking: A Technical Introduction," *Cisco Public White Pap.*, May 2017.

[70] L. Zhang, Y.-C. Liang, and M. Xiao, "Spectrum Sharing for Internet of Things: A Survey," *IEEE Wirel. Commun.*, vol. 26, no. 3, pp. 132–139, Oct. 2018, doi: 10.1109/MWC.2018.1800259.

[71] J. Jeon and T. Cruz, "Cellular Unlicensed Spectrum Technology," in *Encyclopedia of Wireless Networks*, X. S. Shen, X. Lin, and K. Zhang, Eds. Cham: Springer International Publishing, 2018, pp. 1–3.

[72] R. Sharma, V. Vashisht, and U. Singh, "Modelling and simulation frameworks for wireless sensor networks: a comparative study," *IET Wirel. Sens. Syst.*, vol. 10, no. 5, pp. 181–197, Oct. 2020, doi: 10.1049/iet-wss.2020.0046.

[73] Xiaodong Xian, Weiren Shi, and He Huang, "Comparison of OMNET++ and other simulator for WSN simulation," in *2008 3rd IEEE Conference on Industrial Electronics and Applications*, 2008, pp. 1439–1443, doi: 10.1109/ICIEA.2008.4582757.

[74] S. Mittal, "OPNET: An Integrated Design Paradigm for Simulations," *Softw. Eng. An Int. J.*, vol. Vol. 2, no. No. 2, pp. 68–84, Sep. 2012.

[75] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools '08)*, 2008, pp. 1–10, doi: 10.5555/1416222.1416290.

[76] M. Pahlevan and R. Obermaisser, "Evaluation of Time-Triggered Traffic in Time-Sensitive Networks Using the OPNET Simulation Framework," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2018, pp. 283–287, doi: 10.1109/PDP2018.2018.00048.

[77] P. Heise, F. Geyer, and R. Obermaisser, "TSimNet: An Industrial Time Sensitive Networking Simulation Framework Based on OMNeT++," in *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2016, pp. 1–5, doi: 10.1109/NTMS.2016.7792488.

[78] OMNeT++, "OMNeT++." [Online]. Available: https://omnetpp.org/. [Accessed: 26-Nov-2019].

[79] D. Hellmanns, "NeSTiNg - A Network Simulator for Time-sensitive Networking," in *September 2018 Interim Meeting in Oslo, Norway*, 2018.

[80] D. Hellmanns and J. Falk, "ipvs / nesting project · GitLab." Distributed Systems group of IPVS, University of Stuttgart.

[81] INET, "INET Framework." [Online]. Available: https://inet.omnetpp.org/. [Accessed: 26-Nov-2019].
[82] IEEE Standards Association, "IEEE Standard for Local and Metropolitan Area Network--Bridges and Bridged Networks," in *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, 2018, p. pp.1-1993, doi: 10.1109/IEEESTD.2018.8403927.
[83] A. Meddeb, "Internet QoS: Pieces of the puzzle," *IEEE Commun. Mag.*, vol. 48, no. 1, pp. 86–94, Jan. 2010, doi: 10.1109/MCOM.2010.5394035.
[84] IEEE Standards Association, "IEEE Standard for Local and metropolitan area networks-- Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams," in *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, 2010, pp. 1–72, doi: 10.1109/IEEESTD.2010.8684664.
[85] SimuLTE, "SimuLTE - LTE User Plane Simulator for OMNeT++ and INET." [Online]. Available: https://simulte.com/. [Accessed: 26-Sep-2020].
[86] A. B. D. Kinabo, J. B. Mwangama, and A. A. Lysko, "An Evaluation of Broadband Technologies from an Industrial Time Sensitive Networking Perspective," in *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2021, pp. 715–722, doi: 10.1109/ICACCS51430.2021.9441748.
[87] Microsoft, "Windows Subsystem for Linux Documentation | Microsoft Docs," *Microsoft Docs*, 15-Sep-2020. [Online]. Available: https://docs.microsoft.com/en-us/windows/wsl/. [Accessed: 05-Oct-2020].
[88] A. Kili, "How to Install VirtualBox Guest Additions in Ubuntu," *TecMint*, 2019. [Online]. Available: https://www.tecmint.com/install-virtualbox-guest-additions-in-ubuntu/. [Accessed: 13-Mar-2020].
[89] Stack Exchange - Ask Ubuntu Forum, "configure - Error while installing Omnet++ on Ubuntu 16.04 : Cannot find osgEarth - Ask Ubuntu," *Ask Ubuntu*, 2018. [Online]. Available: https://askubuntu.com/questions/1035220/error-while-installing-omnet-on-ubuntu-16-04-cannot-find-osgearth. [Accessed: 30-Mar-2020].
[90] A. Kara and R. Hornig, "Omnet++ 5.4.1 running error on Ubuntu 18.10 - Google Groups," 2019. [Online]. Available: https://groups.google.com/forum/#!topic/omnetpp/t13qEhyyuqI. [Accessed: 27-May-2020].

Appendices

May it be noted that, because of the novel coronavirus situation, the laboratory on campus, as well as all other sections of the University, were closed off. The author was only able to continue with the implementation stage of the work through the following action: the author remotely accessed the desktop machine which was situated on the laboratory in campus from another location in order to access the software of the more powerful lab pc. This desktop machine was favourable mostly because of its processing power, but also its storage capabilities and other resources. This was done through the TeamViewer application, upon which successful remote login meant that the author had to further log into the Linux virtual machine in the Virtualbox program on which this work is based. The writer's progress speed was hampered, but it proved manageable.

A1 Pre-configuring the TSN Environment: Options and Motivation

NeSTiNg is only supported by Linux at present, and so it deems the use of one of the Linux OS distributions. Since the lab machines come pre-installed with Windows OS, it either had to be replaced or to work alongside a Linux OS. Four options existed for this: the first, and least desirable, would be to erase the current profile and reinstall Linux as the host OS for the machine; to install Linux alongside Windows in a dual-boot setup; another is to use a fairly recent development in Windows that allows it to run Linux as a service, termed Windows Subsystem for Linux (WSL); finally, to incorporate virtualisation in order to run Linux as a guest OS inside the Windows host device.

The first option is not ideal because it would customise the machine to be used solely for the efforts of this particular research work and no other coinciding academic activities. This would not be such a bad thing if the research lab did not run sparing of resources. But since this is not the case, and laboratory resources are limited, they need to be used wisely. Pursuing this option would mean erasing all current functions of the Windows PC and removing it from the active directory, which includes it as part of the Department's lab machines. It would not be a trivial task to add it to the network to allow for continued system administration alongside the other Windows desktops, for tasks such as monitoring, running software patches, antivirus and other programs update and/or installation.

The next method of installing Linux alongside Windows is resource-intensive because it essentially demands the splitting of resources between the two host operating systems. On PC start-up, it would offer the user the chance to choose between the two OSes which one to boot. It offers similar trade-offs as in the first option, and so is not desirable.

The third option, WSL, is a tool that would provide a compatibility layer that allows Windows 10 and Windows Server 2019 to run a full Linux/UNIX kernel inside them, as documented in [87]. After its installation, a Linux distribution such as Ubuntu can be downloaded as an

application off the Windows Store, to be used almost like any other program. It similarly incorporates aspects of virtualisation to afford Windows a virtual machine (VM) based solution, but it uses only some of the VM features and so is less resource-intensive than a regular VM program or dual-boot set-up [87]. The writer confirmed with the authors of the NeSTiNg tool that their framework had not been tested with WSL before.

The most favourable option was through virtualisation tactics – the system would run in a virtualised environment and so required the installation of virtualisation software. These programs are more flexible and intuitive in their design, and the resources allotted to their VMs can be easily adjusted as is necessary. If the application is running with too much overhead, the program's GUI allows for the reduction of said resources and even for the definition of constraints to limit the VMs' consumption. Moreover, several VMs can be built and run as needed. One such virtualisation program chosen for this was the Oracle VirtualBox program.

After the installation of VirtualBox, next would be to create a new Linux guest OS and apportioning the required system resources to support it: processing power, memory, and storage, after which other settings were configured. Along with the installation of the OS followed that of Linux Guest Additions for a seamless use of the system. This was mounted as an ISO file to the relevant VM.

From here on, the rest of the development could continue.

A2 Installation, Program Files and Development

The procedure for installing OMNeT, virtualization and other tools for the experiment relevant to Section 4 is detailed here. Along with the relevant files, this section also documents a list of the errors encountered during the installation and development phases, and how they were resolved.

Install procedure:

1) On the Windows machine, the Oracle VM VirtualBox Manager was downloaded and installed.
2) An ISO image of the latest LTS Ubuntu OS version at the time (Ubuntu 18.04 – Bionic Beaver) was obtained from the Ubuntu official site
3) A virtual machine was created using the ISO. Detailed steps can be found on the VirtualBox site
4) Additional settings were configured once the VM was up and running. These are known as VirtualBox Guest Additions.
   They are a collection of device drivers and system applications that aid the user to have a better experience while working in a virtualised environment. Although similar settings come with the Ubuntu (or other OS) ISO image, usually the repository is outdated, and so they do not fit so well with the current VM settings. These VirtualBox Guest Additions ship with the Oracle VirtualBox software, and their install procedure is detailed quite clearly here [88].
   The guest additions help with a number of things [88]:
   - Easy mouse pointer integration
   - Shared clipboard for copying and pasting things both ways between the guest and host systems
   - Drag and drop feature, which also works bi-directionally between the guest and host
   - Seamless Windows feature, which was the most important for this work. It enabled the windows open in the guest machine to automatically adjust to fit the Windows of the host, for a better viewing experience
   - Other features include accelerated video performance and better time synchronisation, to name a few.
5) Next came the task to install the foundation for the simulation environment.
   As highlighted in Fig 14 (Section 4.2), OMNeT++ was first on the list. The prescribed OMNeT version was chosen going by the instructions of the Distributed Systems group of IPVS. Detailed instructions on its installation can be found in its (OMNeT) documentation [78]. Although sometimes these steps do not go according to plan majorly because of unmatched program dependencies, a fact usually stemming from

outdated programs in the Ubuntu repository, which leads to missing prerequisite programs. Secondary assistance came through a source found here [89]

6) The INET version chosen by the Distributed Systems group was used to run its installation.
   Even though OMNeT provides a prompt for downloading the INET library upon its first-time use, sometimes it has to be installed manually, which was the case in this project. Instructions from the INET installation page on its home site help with this [81].
7) Even after successful installation, there may be further errors encountered when launching the OMNeT IDE, prompting the user to check the log files. It was revealed in [90] that "OMNeT++ 5.4.1 is based on Eclipse 4.7.3 and The java 11 broke the compatibility with Eclipse 4.7 which in turn causes the issue with OMNeT++ 5.4.1." The author tried his second workaround for the problem and solution described, and found it cleared the error immediately.
8) The rest of the procedure for installing NeSTiNg itself is as documented in [80].
   The install procedure was replicated to match the steps highlighted therein.

Files

In the development and testing phases, the main files created were as follows:

- VlanEtherHostQWireless.ned, whose file path is: `my_project_workspace/nesting2/src/nesting/node/ethernet/VlanEtherHostQWireless.ned`
- 01_wifi_strict_priority.ini, WiFiTestScenario.ned, 01_wifi_strict_priority_2.ini, WiFiTestScenario2.ned, 01_example_wifi_strict_priority_3.ini, WiFiTestScenario3.ned, 01_example_wifi_strict_priority_4.ini, WiFiTestScenario4.ned, best_effort_only.ini, BestEffort.ned, best_effort_only_2.ini, BestEffort2.ned, minimal_reproducible_example.ini, ReproducibleExample.ned, tester.ini, Tester.ned, wi-fi_example.ini, wi-fi_example.ned, all in the same directory, whose file path is: `my_project_workspace/nesting2/simulations/examples/`
- TestScenarioRouting.xml, TestScenarioRouting_2.xml, TestScenarioSchedule_AllOpen.xml, TestScenarioSchedule_AllOpen_2.xml, TestScenarioSchedule_AllOpen_3.xml, wifi_network.xml, wifi_network_switched.xml, wifi_strict_priority.xml, found in the same directory with the file path: `my_project_workspace/nesting2/simulations/examples/xml/`

Some of the NeSTiNg authors' original files were also modified for testing purposes:

- 01_example_strict_priority.ini, TestScenario.ned, 01_example_strict_priority2.ini, TestScenario2.ned, extraswitch.ini, ExtraSwitch.ned, all in the same directory, whose file path is: `my_project_workspace/nesting/simulations/examples/`
- TestScenarioRouting.xml and TestScenarioSchedule_AllOpen.xml, on the file path: `my_project_workspace/nesting/simulations/examples/xml/`

Other files were reviewed as required for understanding and in the testing. Some of the main ones are:

- 02_example_gating.ini, 03_example_frame_preemption.ini, on the path: `my_project_workspace/nesting2/simulations/examples/`
- TestScenarioSchedule_GatingOn.xml on the path: `my_project_workspace/nesting2/simulations/examples/xml/`

The system further produced results folders to hold the result files generated from the various tests and simulations. These folders are easily differentiated because they are named after the specific simulation they describe. They lie on the path: `my_project_workspace/nesting2/simulations/examples/<results_folder>` and `my_project_workspace/nesting/simulations/examples/<results_folder>`

Errors in Development

Upon building and running the programs developed in the simulator, there were more errors encountered:

1) No such file or directory

   When running one of the example simulations, this error kept popping up. The writer noted that it was picking only the first word of the project workspace folder's name. It stopped at the first delimiter (space). It was resolved by changing the folder name from "my project workspace" to "my_project_workspace". The simulations could run after this.

   It is a common enough error that it pops up in other spheres of Ubuntu as well – having such spaces being replaced instead by an *underscore* takes care of it.

2) Other coding errors are due to improper specification of the modules, parameters, submodules and importing of the used packages themselves. Some are obvious and are picked up by the simulator console. Others are a bit tricky to solve because the INET tutorial examples and framework guides may use a different notation than that in online forums such as StackOverflow. It may be that the INET page is based on an outdated version of INET, but whatever the case, the editors leave too much to assumption, and so the writer has stocked some meaningful pages which helped with many of the issues he was facing:

a. https://stackoverflow.com/questions/54597785/building-node-with-inet-getcontainingnicmodule-nic-module-not-found
   - Contains help on configuring gates, interfaces and their connections for wireless nodes. Also, on mobility, which for wireless hosts usually has to be declared. Although it should be noted this node extends another and is not a fully custom one, so some commands may differ depending on your goal – which was the case for this work. Still, it provides valuable insight into what has changed from the old INET, and what needs to be configured.
b. https://stackoverflow.com/questions/36615266/how-can-i-combine-my-customize-module-with-omnetinets-simple-module
   - May be a bit outdated, but it provides a thorough guide on how to build and customize your own module, which is essentially what NeSTiNg does with its VlanEtherHostQ.ned file, for example. Provides a good reference on what needs to be included. Pay attention as some of the syntax is for older INET and OMNeT++ versions.
c. https://stackoverflow.com/questions/50617868/connecting-wireless-hosts-to-a-standard-host-via-ap-in-omnet
   - Good base to start as well because the scenario is of one which sees a wireless host connected to an access point which in turn is connected to a server: it mixes wireless and wired links, which is what the writer worked on in his simulation environment. Fairly recent. Helps with what needs to be declared in the ".ini" file and also adds onto knowledge on the ".ned" file configuration.
d. The writer had to change the initial interface configuration from "WirelessInterface" to "Ieee80211Interface" in the VlanEtherHostQ.ned file.
   Reason – The writer came to realise that certain configurations come pre-configured with parameters that would otherwise need to be defined in other layouts/configurations.
   For example: another error concerning '.mib' needing to be specified. This was brought on by entering the following for the mac specification with "WirelessInterface" chosen: mac.typename = "Ieee80211Mac". But then you have to specify parameters like 'mib', which the system informs you as an error; to avoid this, the writer resolved to change the mac type to "CsmaCaMac" instead. What this did was bring a complete system, with the simulation finally beginning to run past the initial errors.

   But as soon as it starts transmitting packets from the Controller, once they reach the first switch, the next error comes up: "Upper command 'LLC_OPEN' is not handled".

On reading about this logical link control and searching through the INET framework for possible matches about the error, the writer concluded it has to do with either encapsulation, or with destination mac address not being specified. At this point the author found it easier to change the interface from "WirelessInterface" to "Ieee80211Interface". With this interface you can run the system with mac type as "Ieee80211Mac" without any 'mib' error. And the simulation runs past the LLC_Open error. Until it reaches something else at least.

e. The exact message is: "STA is not associated with an access point", in the console IDE as the simulation is running. It is not an error per se, rather it is something to note of the host in question.
   Reason: In the case of this work, it was missing the definition of the host's management access point address, which is the same as the associated access point's mac address (defined without the need for keyword "mac"). This may be defined for the STA as "wlan[*].mgmt.accessPointAddress". But if the wireless interface definition is incomplete then it will be of no use. So, the full definition of wlan must be defined, as in the next point.

f. For a complete definition of the wireless interface of a custom host or node, say Ieee80211Interface, one needs to indicate first in the gates input, the number of radios. This same number must be defined in the wireless interface section, as pointed out in: https://stackoverflow.com/questions/36615266/how-can-i-combine-my-customize-module-with-omnetinets-simple-module
   Example:
   "wlan[1]: <default("Ieee80211Interface")> like IWirelessInterface {,,,,,,,}", means one wireless interface of the type "Ieee80211Interface".
   Again, this has to be stated in the connections section, but this time the number represents the index or number of the interface in use. For the case above, one interface means starting at zero: wlan[0].

g. Runtime error: "Implicit chunk serialization is disabled to prevent unpredictable performance degradation (you may consider changing the Chunk::enableImplicitChunkSerialization flag or passing the PF_ALLOW_SERIALIZATION flag to peek) -- in module (inet::visualizer::MediumCanvasVisualizer)"
   – This issue was opened in NeSTiNg's Github in March 2019 (issue #8), and closed sometime thereafter (See: https://gitlab.com/ipvs/nesting/-/issues/8). It seems there was an issue with the code; it has been updated to resolve it. Still, the writer couldn't figure out why their INET installation picked up this error even though it was much more recent. The writer used Git commands to pull/merge the proper stable branch with their local folder.

A3 Concerns of Erroneous Results from Memory Limitations

The system's hardware specifications were an initial point of concern; the host system had an internal memory of 8 GB of RAM. Of this, only 2 GB could be safely allotted to the virtual operating system without having a detrimental effect on the performance of the host system that ran the virtual machine. This places a limit on how resource-intensive the virtual machine simulation program can be. However, if the amount of resources required at any one time exceeds a certain pre-set value, the system begs use of "swap memory" to extend the available system resources.

In other words, this technology enables a computer's operating system to provide extra memory to a process or application over and above what the physical RAM can provide. It does this by creating a swap file. Physical memory (RAM) is faster than virtual memory, which is stored on disk. Applications work more smoothly if they are reading from RAM than from disk memory. And if the application or process is running low on RAM as discussed, the operating system will switch out those older unused processes or applications, swapping them out. The exchange happens from real memory transporting or storing them to virtual memory. When they are needed to run again, they can be swapped back in. Meantime the freed real memory can be used to run the current applications that are in need of more memory resources.

With the concept of swap memory in mind, there were concerns that if the simulation process be swapped between memory and disk even just once, it would affect the timing, making it randomised. After a few careful trials it was ascertained that as long as the system variables and other conditions were kept constant, and it was not overloaded, that things would run smoothly. The memory proved sufficient for some initial verification tests, and it was concluded that it could always be extended if the system became too burdened or sluggish to cope in latter stages. Moreover, the experimentation could be repeated for a number of iterations to get some reliable average value so as to defeat any bias or erroneous results spawned from isolated incidents.

B Additional Comments Regarding Suggested Future Work

Algorithm for Time Synchronisation for Future Work

From [14], which details the experiment conducted by Cho et al., this work introduces a simple pseudocode algorithm that can be the basis of future work for the needed global clock synchronisation feature. From the earlier description of the synchronisation mechanism in IEEE 1588 PTP, the following flow can be detailed. It is carried out in two stages.

First stage:

1. Master clock sends sync_request message to slaves
2. Master clock timestamps sync_followup message with transmission time of sync_request
3. Master clock sends sync_followup message to slaves
4. Slave receives sync_request and timestamps its arrival time with the local clock
5. Difference between the timestamps is calculated
6. Value is the sum of offset of slave and message transmission delay

Second stage:

1. Slave clock timestamps delay_request message
2. Slave sends delay_request message to master clock
3. Master clock receives and timestamps delay_request
4. Master clock timestamps delay_response message with arrival of delay_request
5. Master clock sends delay_response to slave
6. Difference between the two timestamps is calculated
7. Value is the slave-to-master delay

Both offset and message delay values are periodically used to correct clocks, and so keep master and slaves synchronised.

C Raw Data from Simulations

Table 11 presents the throughput results (Eq 2) for the number of frames successfully delivered in the three different setups over nine separate cases of increasing bandwidth.

Table 11: Initial Cases

CASES (No of packets transferred over 1 second)																					
	at 11Mbps (Eth = 10Mbps)		*at 36 Mbps (Eth = 10 Mbps)*		*at 36 Mbps (Eth = 100 Mbps)*		*at 54Mbps (Eth = 10Mbps)*		*at 54Mbps (Eth = 100 Mbps)*		*at 54Mbps (Eth = 1000Mbps)*		*at 693.3Mbps (Eth = 10Mbps)*		*at 693.3Mbps (Eth = 100Mbps)*		*at 693.3Mbps (Eth = 1Gbps)*	
	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames
non-TSN alone	NA	**177**	NA	**461**	NA	**417**	NA	**552**	NA	**542**	NA	**544**	NA	**805**	NA	**823**	NA	**855**
TSN alone	**958**	NA	**1958**	NA	**1960**	NA	**2067**	NA	**2068**	NA	**2068**	NA	**2498**	NA	**2500**	NA	**2500**	NA
TSN+non-TSN	860	146	2495	169	1205	1068	2495	169	1111	1421	1640	1165	2496	169	717	2198	1497	1797
	As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:	
non-TSN alone	2.03		5.28		4.77		6.32		6.20		6.23		9.21		9.42		9.78	
TSN alone	2.59		5.29		5.29		5.58		5.59		5.59		6.75		6.75		6.75	
TSN+non-TSN	2.32	1.67	6.74	1.93	3.25	12.22	6.74	1.93	3.00	16.26	4.43	13.33	6.74	1.93	1.94	25.15	4.04	20.57
	3.99		8.67		15.48		8.67		19.26		17.76		8.68		27.09		24.61	

Table 6 in the main text shows a summary of the above data.
Table 12 shows the results for the single access point setup. In order to compare this setup against the one with two access points above it, the same values were used, that is, 1 workstation sending out 1500-byte packets at 12 µs intervals.

Table 12: Initial Cases with 1 Access Point

		After AccessPoint1 removed																	
		CASES (No of packets transferred over 1 second)																	
		at 11Mbps (Eth = 10Mbps)		*at 36 Mbps (Eth = 10 Mbps)*		*at 36 Mbps (Eth = 100 Mbps)*		*at 54Mbps (Eth = 10Mbps)*		*at 54Mbps (Eth = 100 Mbps)*		*at 54Mbps (Eth = 1000Mbps)*		*at 693.3Mbps (Eth = 10Mbps)*		*at 693.3Mbps (Eth = 100Mbps)*		*at 693.3Mbps (Eth = 1Gbps)*	
		TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames
	non-TSN alone	NA	**queue error**	NA	**queue error**	NA	**queue error**	NA	**queue error**	NA	**queue error**	NA	**1354**	NA	**queue error**	NA	**queue error**	NA	**2363**
	TSN alone	**958**	NA	**1958**	NA	**1960**	NA	**2067**	NA	**2068**	NA	**2068**	NA	**2498**	NA	**2500**	NA	**2500**	NA
	TSN+non-TSN	queue error	queue error	queue error	queue error	queue error	queue error	queue error	queue error	queue error	queue error	13	1345	queue error	queue error	queue error	queue error	17	2352
		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:	
	non-TSN alone		#VALUE!		#VALUE!		#VALUE!		#VALUE!		#VALUE!		15.5		#VALUE!		#VALUE!		27.04
	TSN alone	2.59		5.29		5.29		5.58		5.59		5.59		6.75		6.75		6.75	
	TSN+non-TSN	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	0.04	15.39	#VALUE!	#VALUE!	#VALUE!	#VALUE!	0.05	26.92
		#VALUE!		#VALUE!		#VALUE!		#VALUE!		#VALUE!		15.43		#VALUE!		#VALUE!		26.96	

The queue error led to invalid values for certain fields. The summary of the above follows in Table 13. Invalid values are shown as "#VALUE!".

Table 13: Summary

	Wireless	non-TSN alone	TSN alone	TSN w nonTSN	nonTSN w TSN	TSN+nonTSN
Mbps	**11**	#VALUE!	2.59	#VALUE!	#VALUE!	#VALUE!
	36	#VALUE!	5.29	#VALUE!	#VALUE!	#VALUE!
	36	#VALUE!	5.29	#VALUE!	#VALUE!	#VALUE!
	54	#VALUE!	5.59	#VALUE!	#VALUE!	#VALUE!
	693.3	#VALUE!	6.75	#VALUE!	#VALUE!	#VALUE!

The Queue error in question is a runtime error, reproduced here: `<!> txQueue length exceeds 10000 -- this is probably due to a bogus app model generating excessive traffic (or if this is normal, increase txQueueLimit!) -- in module (inet::EtherMacFullDuplex) WiFiTestScenario2.workstation1.eth.mac (id=400), at t=0.132996s, event #130133`

Table 14: Critical Intervals for Queue Error

Measured Critical time intervals, below which the queue error results			
11 Mbps	36 Mbps	54 Mbps	693.3 Mbps
10 Mbps Eth = 93 µs			
100 Mbps Eth = 56 µs			
1 Gbps Eth= 11 µs			

The scenarios in Table 7 in the main text were run to understand the reason behind the losses occurring at the access point.

For the sake of brevity, only one of the scenarios will be fully laid out (Table 15), while a brief, but comprehensive summary of all of them will follow in the manner of Table 17 and Table 18.

Table 15: Scenario 5

		Scenario 5							
		at 11Mbps (Eth = 10Mbps)			*at 54Mbps (Eth = 10Mbps)*			*at 693.3Mbps (Eth = 1Gbps)*	
		TSN frames	non-TSN frames		TSN frames	non-TSN frames		TSN frames	non-TSN frames
non-TSN alone		NA	199		NA	199		NA	199
			199			199			199
			199			199			199
	Mean		199.0			199			199
	Loss		1.0			1			1
	Std dev		0.0			0			0
TSN alone		958	NA		2067	NA		2500	NA
		958			2067			2500	
		958			2067			2500	
	Mean	958.0			2067			2500	
	Loss	1542.0			433.0			0.0	
	Std dev	0			0			0	
TSN+non-TSN		704	147		1839	171		2500	199
		704	147		1839	171		2500	199
		704	147		1839	171		2500	199
	Mean	704	147		1839	171		2500	199
	Loss	1796	53		661	29		0	1
	Std dev	0	0		0	0		0	0
		As throughput, Mbps:			As throughput, Mbps:			As throughput, Mbps:	
	non-TSN alone		2.28			2.28			2.28
	TSN alone	2.59			5.58			6.75	
	TSN+non-TSN	1.90	1.68		4.97	1.96		6.75	2.28
		3.58			6.92			9.03	

Table 16 shows the calculated values for the amount of time it is expected to take to transmit one of each kind of packet, with the given sizes and link speeds.

Table 16: Expected Duration for Frame Transmission

				Speed, Mbps	sec per 1 packet	*sec per 10k packets*
1	TSN	354	bytes, Frame duration in Wireless	11	2.57E-04	*2.575*
	non-TSN	1400	bytes, Frame duration in Eth	10	1.12E-03	*11.200*
2	TSN	354	bytes, Frame duration in Wireless	54	5.24E-05	*0.524*
	non-TSN	1400	bytes, Frame duration in Eth	10	1.12E-03	*11.200*
3	TSN	354	bytes, Frame duration in Wireless	693.3	4.08E-06	*0.041*
	non-TSN	1400	bytes, Frame duration in Eth	1000	1.12E-05	*0.112*

The seconds per packet is calculated in this manner (Eq 3):

$$Time\ per\ packet = \frac{bytes}{packet} \times \frac{bits}{byte} \div Link\ speed\ in \frac{bits}{second}$$

Eq 3

For 10000 packets, the result from Eq 3 was multiplied by 10000 in kind.

In Table 17 that follows, the fully summarised information on the numbers of packets lost in the scenarios above can be found. It has had to be truncated for length, with the portion cut off appended below it. The yellow blocks indicate the fields in the scenarios where errors were expected in real life, according to the calculations made in Table 16. This means that, if a field was yellowed out, the chosen send intervals for the frames were not within the range of realistic values; they were sent out too fast. Scenario 5 was noted to have all values within range.

The needed throughput was calculated as shown in Eq 4:

$$Needed\ throughput = Total\ no.of\ frames \times frame\ size\ in\ \frac{bytes}{frame} \times \frac{bits}{byte} \div 1000000$$

Eq 4

The percentages shown are a ratio of the packets lost to the total sent out, specifically for Case 3.

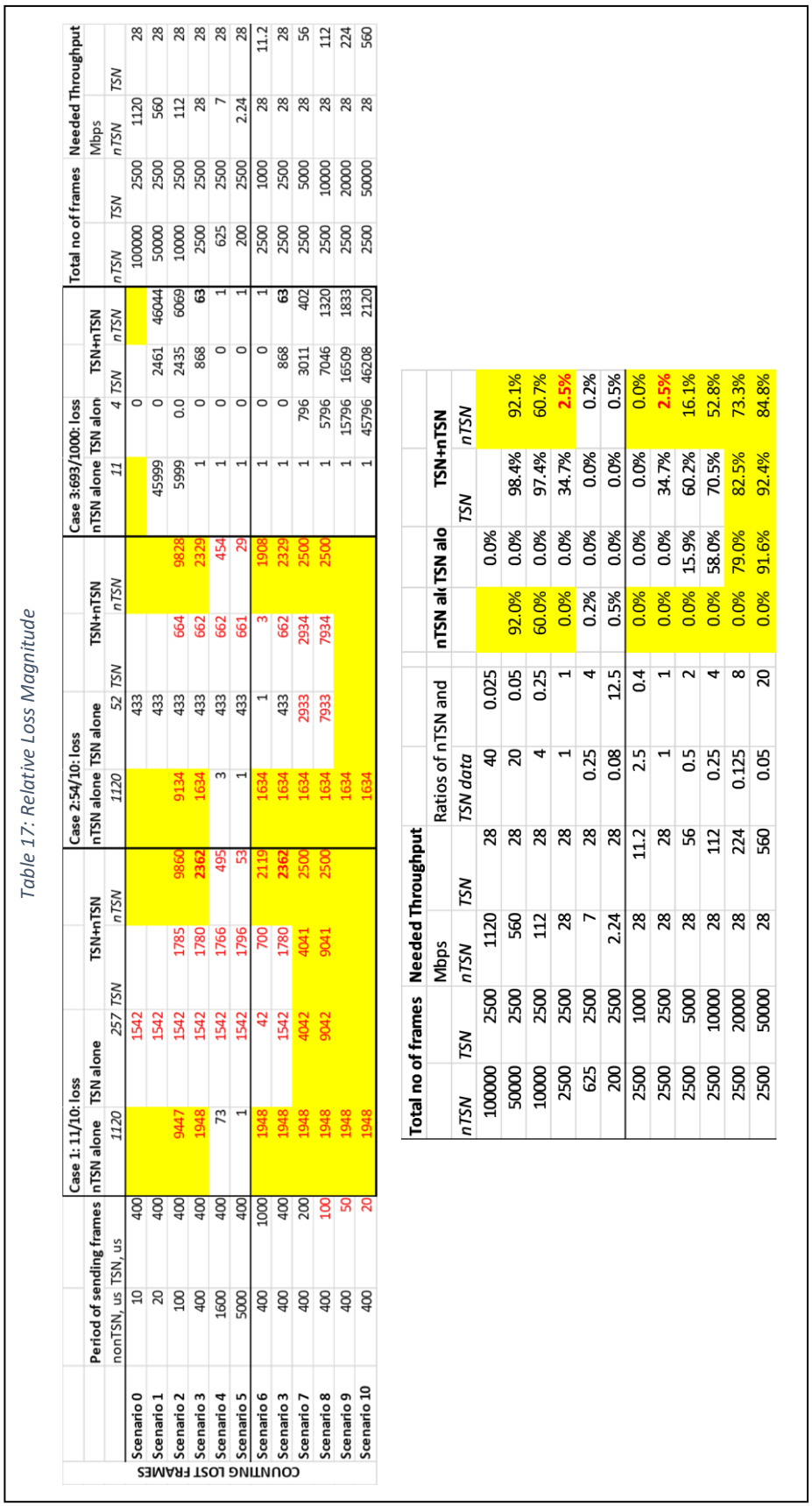

Table 17: Relative Loss Magnitude

COUNTING LOST FRAMES	Period of sending frames		Case 1: 11/10: loss				Case 2:54/10: loss				Case 3:693/1000: loss				Total no of frames		Needed Throughput	
			nTSN alone	TSN alone		TSN+nTSN	nTSN alone	TSN alone		TSN+nTSN	nTSN alone	TSN alon		TSN+nTSN			Mbps	
	nonTSN, us	TSN, us	*1120*		*257 TSN*	*nTSN*	*1120*	*52 TSN*		*nTSN*	*11*	*4 TSN*		*nTSN*	*nTSN*	*TSN*	*nTSN*	*TSN*
Scenario 0	10	400			1542			433				0			100000	2500	1120	28
Scenario 1	20	400			1542			433			45999	0	2461	46044	50000	2500	560	28
Scenario 2	100	400	9447		1542	1785	9134	433	664	9828	5999	0.0	2435	6069	10000	2500	112	28
Scenario 3	400	400	1948		1542	1780	1634	433	662	2329	1	0	868	63	2500	2500	28	28
Scenario 4	1600	400	73		1542	1766	3	433	662	454	1	0	0	1	625	2500	7	28
Scenario 5	5000	400	1		1542	1796	1	433	661	29	1	0	0	1	200	2500	2.24	28
Scenario 6	400	1000	1948		42	700	1634	1	3	1908	1	0	0	1	2500	1000	28	11.2
Scenario 3	400	400	1948		1542	1780	1634	433	662	2329	1	0	868	63	2500	2500	28	28
Scenario 7	400	200	1948		4042	4041	1634	2933	2934	2500	1	796	3011	402	2500	5000	28	56
Scenario 8	400	100	1948		9042	9041	1634	7933	7934	2500	1	5796	7046	1320	2500	10000	28	112
Scenario 9	400	50	1948				1634				1	15796	16509	1833	2500	20000	28	224
Scenario 10	400	20	1948				1634				1	45796	46208	2120	2500	50000	28	560

Total no of frames		Needed Throughput		Ratios of nTSN and		nTSN al	TSN alo	TSN+nTSN	
		Mbps							
nTSN	*TSN*	*nTSN*	*TSN*	*TSN data*			*TSN*	*nTSN*	
100000	2500	1120	28	40	0.025		0.0%		
50000	2500	560	28	20	0.05	92.0%	0.0%	98.4%	92.1%
10000	2500	112	28	4	0.25	60.0%	0.0%	97.4%	60.7%
2500	2500	28	28	1	1	0.0%	0.0%	34.7%	2.5%
625	2500	7	28	0.25	4	0.2%	0.0%	0.0%	0.2%
200	2500	2.24	28	0.08	12.5	0.5%	0.0%	0.0%	0.5%
2500	1000	28	11.2	2.5	0.4	0.0%	0.0%	0.0%	0.0%
2500	2500	28	28	1	1	0.0%	0.0%	34.7%	2.5%
2500	5000	28	56	0.5	2	0.0%	15.9%	60.2%	16.1%
2500	10000	28	112	0.25	4	0.0%	58.0%	70.5%	52.8%
2500	20000	28	224	0.125	8	0.0%	79.0%	82.5%	73.3%
2500	50000	28	560	0.05	20	0.0%	91.6%	92.4%	84.8%

Further calculations were excluded for brevity. After these loss figures, the relative usage of the network's capacity was obtained, as in Table 18.

Table 18: Relative Network Capacity Usage

REL. USAGE OF CH. CAPACITY

	Period of sending frames		Case 1: 11/10: loss %				Case 2:54/10: loss %				Case 3:693/1000: loss			
			nTSN alone	TSN alone		TSN+nTSN	nTSN alone	TSN alone		TSN+nTSN	nTSN alone	TSN alone		TSN+nTSN
	nonTSN, us	TSN, us	*1120*	*257*	*TSN*	*nTSN*	*1120*	*52*	*TSN*	*nTSN*	*11*	*4*	*TSN*	*nTSN*
Scenario 0	10	400		254.55%				51.85%				4.04%		
Scenario 1	20	400		254.55%				51.85%			56.00%	4.04%	84.81%	84.81%
Scenario 2	100	400	1120.00%	254.55%	1400.00%	1400.00%	1120.00%	51.85%	1400.00%	1400.00%	11.20%	4.04%	20.19%	20.19%
Scenario 3	400	400	280.00%	254.55%	560.00%	560.00%	280.00%	51.85%	560.00%	560.00%	2.80%	4.04%	8.08%	8.08%
Scenario 4	1600	400	70.00%	254.55%	350.00%	350.00%	70.00%	51.85%	350.00%	350.00%	0.70%	4.04%	5.05%	5.05%
Scenario 5	5000	400	22.40%	254.55%	302.40%	302.40%	22.40%	51.85%	302.40%	302.40%	0.22%	4.04%	4.36%	4.36%
Scenario 6	400	1000	280.00%	101.82%	392.00%	392.00%	280.00%	20.74%	392.00%	392.00%	2.80%	1.62%	5.65%	5.65%
Scenario 3	400	400	280.00%	254.55%	560.00%	560.00%	280.00%	51.85%	560.00%	560.00%	2.80%	4.04%	8.08%	8.08%
Scenario 7	400	200	280.00%	509.09%	840.00%	840.00%	280.00%	103.70%	840.00%	840.00%	2.80%	8.08%	12.12%	12.12%
Scenario 8	400	100	280.00%	1018.18%	1400.00%	1400.00%	280.00%	207.41%	1400.00%	1400.00%	2.80%	16.15%	20.19%	20.19%
Scenario 9	400	50	280.00%				280.00%				2.80%	32.31%	36.35%	36.35%
Scenario 10	400	20	280.00%				280.00%				2.80%	80.77%	84.81%	84.81%

For the wired case, experiments were performed for different scenarios shown in Table 8, in the main text.

As in Table 15, only one scenario is fully laid out in Table 19 below.

Table 19: Scenario 4 (wired)

	Scenario 4					
	at Eth = 10Mbps		*Eth = 100Mbps*		*at Eth = 1Gbps*	
	TSN frames	non-TSN frames	TSN frames	non-TSN frames	TSN frames	non-TSN frames
TSN+non-TSN	864	0	4998	3677	5000	4999
	864	0	4998	3677	5000	4999
	864	0	4998	3677	5000	4999
Mean	864	0	4998	3677	5000	4999
Loss	4136	5000	2	1323	0	1
Std dev	0	0	0	0	0	0
	As throughput, Mbps:		As throughput, Mbps:		As throughput, Mbps:	
TSN+non-TSN	9.23	0.00	53.38	39.27	53.41	53.40
	9.23		92.66		106.80	

Similarly, the loss results as those in Table 17 can be calculated, seen in Table 9 in the main text.

For the experiments that mimicked these last three in Table 8, Table 19, and Table 9, in the implementation with two access points connected on one end to *switchB*, those raw results are seen in Table 20, Table 21, and Table 10.

Table 20 similarly captures the different scenarios.

Table 20: Scenarios (wireless: 2 APs, on one end)

SCENARIOS			
	Default packet size and send interval		
0		period, µs	size, bytes
	non-TSN	12	1500
	TSN	400	354
	From this scenario onwards, same packet size and period		
		period, µs	size, bytes
1	non-TSN & TSN	1000	1400
2	non-TSN & TSN	500	1400
3	non-TSN & TSN	300	1400

4	non-TSN & TSN	200	1400
5	non-TSN & TSN	100	1400
6	non-TSN & TSN	50	1400
7	non-TSN & TSN	30	1400
8	non-TSN & TSN	20	1400

Only one scenario is fully laid out in Table 21 below.

Table 21: Scenario 4 (wireless: 2 APs, on one end)

		Scenario 4					
		Eth = 100Mbps				*at Eth = 1Gbps*	
		TSN frames	non-TSN frames			TSN frames	non-TSN frames
TSN+non-TSN		2362	2527			2363	2372
		2362	2527			2363	2372
		2362	2527			2363	2372
	Mean	2362	2527			2363	2372
	Loss	2638	2473			2637	2628
	Std dev	0	0			0	0
		As throughput, Mbps:				As throughput, Mbps:	
	TSN+non-TSN	25.23	26.99			25.24	25.34
		52.22				50.58	

Again, the loss results as in Table 17 and Table 9 can be calculated, shown in Table 10 in the main text.