AN EXPERIMENTAL ANALYSIS OF A CONVERGED TSN-4G NETWORK FOR REAL-TIME APPLICATIONS

Ekpenyong Esu Ekpenyong
eeg21003@student.mdu.se

Abdul Hannan Khalid
akd21001@student.mdu.se
Examiner: Examiner_Name
Muhammad Ashjaei

Supervisors: Supervisor_Name
Zenepe Satka, Saad Mubeen

Company supervisor: John Lundbäck,
Arcticus System, Järfälla, Sweden
Table of Contents

1. Introduction .............................................................................................................. 2
   1.1. Motivation ............................................................................................................ 2
   1.2. Problem Formulation ............................................................................................ 3
   1.3. Expected Outcomes ............................................................................................... 3
   1.4. Thesis Outlines ..................................................................................................... 3

2. Background .................................................................................................................. 4
   2.1. Embedded Systems ............................................................................................... 4
   2.2. Real-Time Embedded Systems .............................................................................. 4
   2.3. End-To-End Latency .............................................................................................. 5
   2.4. Real-Time Operating systems
       2.4.1 Ubuntu Operating System ............................................................................. 6
   2.5. Switched Ethernet .................................................................................................. 6
   2.6. Time-Sensitive Networking (TSN)
       2.6.1 Time Synchronization .................................................................................... 7
       2.6.2 TSN Traffic Classes ....................................................................................... 7
       2.6.3 Scheduling Mechanism .................................................................................. 7
   2.7. 4G-LTE .................................................................................................................. 9
   2.8. Hardware Considerations ..................................................................................... 9
   2.9. Integration of Wired and Wireless Network ........................................................... 10

3. Related Work .............................................................................................................. 11
   3.1. Wired TSN ............................................................................................................ 11
   3.2. 4G LTE ............................................................................................................... 11
   3.3. Integration of TSN with Predictable Applications .................................................. 12

4. Research Method ....................................................................................................... 15
   4.1. System Development Research Method ............................................................... 15
   4.2. Platform-Based Development .............................................................................. 16
   4.3. Discussion ............................................................................................................ 16

5. Prototype Design ....................................................................................................... 17
   5.1. Communication and Processing Latency over TSN-4G ........................................... 17
       5.1.1 Block Diagram ............................................................................................... 18
       5.1.2 Code Implementation ..................................................................................... 20
   5.2. Communication Latency over TSN ....................................................................... 24
       5.2.1 Block Diagram ............................................................................................... 24
   5.3. Communication Latency over 4G .......................................................................... 25
       5.3.1 Block Diagram ............................................................................................... 25

6. Experimental Evaluation ............................................................................................ 27
   6.1. Experimental Setup .............................................................................................. 27
       6.1.1 Single Board Computer .................................................................................. 27
       6.1.2 Raspberry Pi Camera Module ........................................................................ 27
       6.1.3 ZeroTier .......................................................................................................... 28
       6.1.4 TSN Switch ...................................................................................................... 28
       6.1.5 4G Module ........................................................................................................ 29
       6.1.6 Ethernet Cable .................................................................................................. 29
       6.1.7 Preempt Real-Time Patch ............................................................................... 29
       6.1.8 Traffic Characteristics ..................................................................................... 29
   6.2. Experiments .......................................................................................................... 30
       6.2.1 Scenario 1: Communication between Node A and Node B by sending data frames over TSN ......................................................................................... 30
       6.2.2 Scenario 2: Communication between node B and node C by sending data frames over 4G ................................................................................................. 30
6.2.3 Scenario 3: Communication between node A, node B, and node C over an integrated network of 4G and TSN

6.3 Results

6.3.1 Scenario 1

6.3.2 Scenario 2

6.3.3 Scenario 3

6.4 Discussion

7. Threats to Validity

7.1 Establishing Communication Channel

7.2 Data Transmission

7.3 Data Translation

7.4 Single Board Computer

7.5 TSN Switch

8 Conclusion

9 Future Work

10 Acknowledgments

References

Appendix A Appendix 1

Appendix B Appendix 2

Appendix C Appendix 3
Abstract

This thesis conducts an experimental analysis of a converged TSN-4G network for real-time industrial applications, exploring the utilization of both real-time operating systems and non-real-time operating systems. The primary aim is to evaluate the feasibility and challenges associated with integrating Ethernet-based Time-Sensitive Networking (TSN) over Ethernet with 4G LTE wireless technology, with the goal of establishing robust and low-latency communication for industrial automation. While TSN ensures that data is sent and received at predictable times and in a synchronized manner, its reliance on wired connectivity introduces flexibility constraints. The integration of TSN with widely available 4G networks is seen as a potential solution to overcome this limitation while retaining the essential benefits of TSN. A testbed, consisting of Linux-based nodes interconnected through a TSN switch and 4G links, was developed. Three scenarios—TSN-only, 4G-only, and converged TSN-4G—were assessed, with end-to-end latency measurements conducted under various operating systems and network configurations. Critical findings emphasize the pivotal role of real-time operating systems in achieving low latency compared to standard Ubuntu. While the TSN network displayed consistent performance, 4G latency exhibited variability. The introduction of clock synchronization between networks introduced jitter. The integrated TSN-4G network, leveraging real-time operating systems (Ubuntu with RT_Patch), achieved reasonable latencies, with a minimum end-to-end latency of 80.86 ms, a maximum of 106.76 ms, and an average of 94.30 ms, indicating initial feasibility. In contrast, the use of standard Ubuntu resulted in a minimum end-to-end latency of 165.34 ms, a maximum of 215.32 ms, and an average of 189.80 ms. This thesis provides empirical insights into the integration of TSN with 4G networks for real-time industrial communication. The central challenge lies in balancing the deterministic nature of TSN with the non-predictable characteristics of wireless 4G. The study establishes a foundational understanding, paving the way for future research on converged TSN-cellular networks for Industry 4.0 applications that demand reliable and low-latency connectivity.
1. Introduction

The industrial world has undergone three major transformations since the 1800s, each characterized by a revolutionary new technology that fundamentally changed the way goods were produced and work was performed. Currently, the world is experiencing the Fourth Industrial Revolution, commonly referred to as Industry 4.0, which leverages smart technologies to transform the automation, monitoring, and analysis of supply chains. The foundation of Industry 4.0 is the Industrial Internet of Things (IIoT) and cyber-physical systems, which are intelligent and autonomous systems that use computer-based algorithms to monitor and operate physical objects like machinery, robots, and vehicles. To exchange information between these devices, several network connections have been established, including the use of Ethernet-based networks and Field buses [1, 2].

However, many of these networks are unable to handle real-time systems, as their accuracy depends not only on the system’s logical computation but also on its ability to produce the correct response in a predictable amount of time [3]. To address this limitation, the Time-Sensitive Networking (TSN) was developed through the collaboration between industry and academia [4, 5]. TSN offers low latencies, time synchronization, and dedicated resources, making it an ideal solution for supporting real-time connections in smart factories and real-time applications in Industry 4.0. While TSN is ideal for onboard communication [6], wired networks inherently lack support for mobility and flexibility [7]. Also, it can be difficult to establish in locations where cables are inaccessible. To address this challenge, industrial automation is integrating TSN with various wireless technologies such as WiFi, 4G, and 5G [8, 9, 10]. 4G provides ubiquitous wireless connectivity through technologies like LTE but lacks native support for real-time communication.

While the convergence of TSN and 4G has been discussed at a conceptual level [11, 9], existing studies have limitations: primarily theoretical evaluations and in most cases sole reliance on simulations of TSN-4G networks which do not capture the practical challenges in implementation. Lack of analysis using real-time operating systems and limited experiments focused on 5G over 4G [12, 13] [14, 11]. There is a need for in-depth testbed experiments that analyze metrics like end-to-end latency demonstrating the feasibility of supporting real-time industrial applications over converged TSN-4G networks. The integration of 4G and TSN with a real-time operating system has the potential to revolutionize the way industrial processes are implemented and controlled, opening up new applications and enhancing the performance and dependability of current systems. This thesis aims to integrate these technologies into a single network and assess its end-to-end latencies through experimental analysis. The ultimate goal is to demonstrate the feasibility of utilizing TSN and 4G networks in industrial automation and to provide insights into the benefits and limitations of this integration.

1.1. Motivation

In recent years, the concept of Industry 4.0 and the transition towards smart factories have brought about a demand for more sophisticated industrial communication networks. Time-sensitive networking (TSN) has emerged as a critical technology, offering features such as precise time synchronization, low latency, and deterministic behavior that are crucial for various industrial applications. However, TSN’s current limitation lies in its lack of support for mobility and flexibility due to its reliance on wired Ethernet connections. Integrating TSN with wireless networks holds the potential to overcome this limitation while retaining its key benefits [15]. Time-sensitive networking (TSN) is designed to provide deterministic communication and time synchronization within industrial networks [15]. TSN’s capabilities make it well-suited for real-time applications, but it faces challenges when it comes to accommodating mobile devices or dynamically changing network topologies. One promising solution to address this limitation is to integrate TSN with wireless communication technologies. Wireless communication offers mobility and flexibility, allowing devices to move within the network while maintaining connectivity. Among the available wireless options, 4G LTE (Long-Term Evolution) stands out as an attractive candidate to complement TSN. 4G LTE offers mobility support, reasonable latency in the order of tens of milliseconds, and widespread deployment. While 5G technology also holds promise for industrial applications, it is still in the process of being rolled out and might be implemented as private networks with limited user control. In contrast, 4G LTE networks offer a more mature and ubiquitous option that al-
allows for more configurable and customizable deployments [16, 17]. While TSN is deterministic, 4G performance can vary due to factors like signal interference. Evaluating the performance of an integrated TSN-4G network is therefore important and quantifying end-to-end latency under different conditions provides insight into the feasibility of such converged networks for industrial use cases, especially in real-time monitoring and control. This thesis explores the challenges in enabling a converged TSN-4G network. It utilizes an experimental approach to evaluate end-to-end latency across such an integrated network. The impact of interference and non-determinism introduced by the 4G segment is analyzed. This provides a baseline for future research into converged wired and wireless networks for industrial communications [18].

1.2. Problem Formulation

In the context of contemporary industry and the advent of Industry 4.0, the significance of real-time communication cannot be overstated. The intricate interplay of diverse real-time applications, whether they fall under the category of hard real-time or soft real-time, underscores the need for a robust mechanism to convey data. Particularly in critical hard real-time situations, the repercussions of failing to meet deadlines can be severe. The impetus behind this thesis lies in addressing the imperative to facilitate real-time data transmission across industrial applications, encompassing various data types such as audio, video, instructions, and other forms of industrial communication. The central objective of this study is to conduct a thorough experimental analysis of a converged TSN-4G network tailored for real-time applications. Within this context, the following research questions are explored:

- RQ 1: What are the challenges in integrating TSN with a 4G network to fulfill the demanding network communication of real-time applications?
- RQ 2: How can TSN be effectively integrated with a 4G network to achieve predictable low-latencies for real-time applications?

In pursuit of answers to these questions, a comprehensive experimental analysis will be executed on a tangible test bed, replicating the conditions of a converged TSN-4G network. The outcomes of this analysis will illuminate the feasibility, challenges, and benefits of intertwining TSN and 4G networks in the domain of industrial automation.

1.3. Expected Outcomes

An extensive investigation of the functionality of a converged TSN-4G network for real-time applications is the predicted output of this thesis. It is also envisaged that the results will lead to the convergence of two different networks: TSN and 4G. The thesis solely considers the Schedule Traffic (ST) type of time-sensitive traffic on TSN. By combining TSN and 4G, this thesis evaluates the performance of the network in terms of latencies by calculating the overall end-to-end transmission and processing latencies. We will also take into account the challenges of 4G and TSN integration.

1.4. Thesis Outlines

The flow of this thesis is organized as follows. In Section 2 the relevant background is explained. Section 3 explains the overview of current research relevant to this thesis. In Section 4 research method is introduced. Section 5 explains the system model as well as the prototype design developed in accordance with the research method, while the implementation part is specified in Section 6. In Section 7 the experimental analysis and results from implementation are mentioned. Section 8 and Section 9, converse about the limitations of the work done in the thesis and conclusions drawn on the basis of limitations and presented results respectively. In the end, section 10 suggests an empirical approach for future work in this domain.
2. Background

2.1. Embedded Systems

Nearly all of the products we use, including alarm clocks, Personal digital assistants (PDAs), mobile phones, and automobiles, are controlled by embedded systems. Embedded systems use more than 99% of the microprocessors made today, and as of late, the world’s installed base of embedded systems has surpassed that of people. The limitations on functionality, performance, low energy consumption, reliability, cost, and time-to-market are getting tighter as embedded systems complexity increases at a rapid rate. Consequently, the work of creating such systems is becoming both more crucial and challenging. In order to successfully manage the complexity of embedded systems meet the restrictions imposed by the application domain, accelerate time-to-market, and lower development and manufacturing costs, new automated design optimization approaches are required [19].

Microprocessor-based systems that are built for a specific purpose, as opposed to general-purpose computers, and embedded into other larger devices with the aim of controlling them, are called embedded systems [20]. They are used in other devices or goods to carry out a specific task. They are made to work in situations where the system must react to inputs and events in a predictable and brief amount of time, or in real-time contexts. Numerous products, including consumer electronics, autos, medical equipment, and industrial automation, use embedded systems.

The typical components of embedded systems hardware, which is application-specific hardware, include a CPU, memory, and peripherals. Embedded processors come in single-core and multi-core varieties and their selection is based on factors such as price, performance, and power consumption. Memory for embedded systems is typically in the form of EEPROM, RAM, or ROM and is used to store data as well as the software that will run on the system. Memory in embedded systems is also governed by software and affects how software is subsequently developed. In embedded systems, peripherals (such as binary outputs, serial outputs, converters, and displays) serve primarily to connect the embedded system to its surroundings [20, 21]. There are a number of issues that need to be resolved in the design and development of embedded systems, such as constrained computational resources, predictable and dependable network communication, memory accessibility, the requirement for real-time performance, and the need for robustness and reliability. The usage of specialized microprocessors, real-time operating systems, and software development approaches are only a few examples of the hardware and software design strategies that embedded systems designers combine to address these difficulties [22, 23].

The creation of high-performance, low-power embedded systems for IoT (Internet of Things) applications has drawn more attention in recent years. In order to overcome the difficulties encountered in the creation of embedded systems, new technologies, and methodologies have been developed. Examples include the usage of heterogeneous computing platforms, edge computing, and cloud-based systems. Additionally, combining embedded systems with other technologies, like machine learning and artificial intelligence, creates new opportunities for creating intelligent and smart systems for Industry 4.0. These systems have the capacity to instantly process and analyze significant volumes of data and make judgments based on the information they gather. This has the potential to increase productivity and accuracy in a variety of industries, including manufacturing, healthcare, and transportation.

2.2. Real-Time Embedded Systems

In the real-time embedded system, the system has to deliver the correct response with timing predictable behaviour. A system is said to be timing predictable for a given system model and a set of assumptions if it is possible to show, prove, or demonstrate that timing requirements specified on the system will be satisfied when the system is executed [24]. Real-time systems have two categories, soft real-time systems, and hard real-time systems. In hard real-time systems missing a deadline can result in the system’s failure and some extreme circumstances cost life. Whereas, in soft real-time systems, some deadlines can be missed. An example of a hard real-time system is a control system of airbags in a car that has to be inflated within milliseconds after the car crash to prevent life-threatening events. In hard real-time systems, the level of endurance is very low or sometimes zero [25].
2.3. End-To-End Latency

In real-time networked systems, end-to-end latency is a critical performance metric that can be decomposed into two primary components: processing latency and communication latency. The end-to-end latency is the total time required for a data frame or packet to traverse from the source node, undergo processing and communication latency at intermediate nodes, and reach the destination node [26]. Processing latency refers to the time consumed by a node to process and handle a received frame, including tasks such as packet inspection, queuing, scheduling, and application-specific operations [27]. This latency component is influenced by the computational capabilities of the nodes and the complexity of the processing tasks involved. Communication latency, on the other hand, is the time required for a frame to move from one node to another on the network medium, wireless or wired [27]. It is affected by factors such as the physical distance between nodes, propagation delays, queuing delays at intermediate network elements, and the overhead of protocol encapsulation and decapsulation. The end-to-end latency is the cumulative sum of the processing latencies at each node along the path and the communication latencies between these nodes [26]. In integrated Time-Sensitive Networking (TSN) and 4G mobile networks, the end-to-end latency can be impacted by the variable delays introduced by the wireless radio link, the additional protocol overhead of TSN and 4G standards, and the potential for increased processing latency due to the complexity of converging these technologies. In addition to processing and communication latencies, end-to-end latency prediction in real-time networked systems considers other types of delays, notably age delay and reaction delay, which are prevalent in industries like automotive manufacturing. These delays, highlighted in studies such as [28, 29, 30], contribute to the overall time required for establishing a viable schedule within real-time embedded systems. This scheduling process aims to ensure that all tasks within the system meet their designated deadlines and adhere to specified constraints. By accounting for these various delays, network planners and engineers can better anticipate and manage latency, ultimately enhancing the system’s ability to fulfill stringent timing requirements in critical applications. Minimizing both processing latency and communication latency is crucial for meeting the stringent end-to-end latency requirements of real-time applications, such as motion control in industrial automation systems, which may demand latencies in millisecond or even sub-millisecond levels. Achieving such low end-to-end latencies in integrated TSN-4G networks requires careful design considerations, including optimized network architectures, efficient resource allocation strategies, and deterministic processing mechanisms at network nodes.

2.4. Real-Time Operating systems

An operating system is a system program that acts as an interface between user-installed applications and the hardware of a computer. They organize effective and proper use of the computer resources and make the computer system simple to use, respectively. An operating system’s four primary functions are process management, memory management, input/output (I/O) management, and inter-process communication and synchronization. Hence, the process creation, loading, and execution control, as well as the interaction of the process with signal events, process monitoring, CPU allocation, and process termination, are all handled by the process management component. Process protection, data exchange protocols, deadlock and livelock detection, synchronization, and coordination are only a few of the topics covered by inter-process communication. Services for file creation, deletion, repositioning, and protection are part of memory management. I/O management deals with read, write, and relocate programs as well as request and release subroutines for a number of peripherals [31]. Real-Time Operating Systems (RTOS) are customized operating systems created to handle real-time computing applications. These operating systems are utilized in a variety of industries, such as industrial control systems, aerospace, defense, and automotive. It offers a wide range of functions, including task management, time management, input and output control, memory management, synchronization, and inter-task communication. Reliability, predictability, performance, compactness, and scalability are just a few of the numerous benefits of RTOS [32].

The capacity to ensure a timing predictable and constant reaction time for system tasks and events is what distinguishes a real-time operating system. Real-time tasks are given the necessary processing resources in a predictable and brief amount of time thanks to the employment of
specialized scheduling algorithms that allocate system resources in this way. The need for a short and compact code size is one of the most critical design factors for real-time operating systems since it enables the operating system to function on embedded systems with constrained memory and computing capabilities. Real-time operating systems must also be extremely dependable and strong since they are frequently employed in safety-critical applications where a failure could have negative effects \[32\]. The usage of real-time communication protocols, such as the Time-Sensitive Networking (TSN) standard, which enables real-time communication between various hardware devices and systems, is another crucial component of real-time operating systems. This is crucial for real-time embedded systems because it makes it possible to coordinate a variety of devices and systems, ensuring that real-time activities and events are carried out consistently and timing predictably \[33\]. The fact that real-time operating systems are continually developing and making new strides in order to meet the escalating needs and difficulties of real-time applications is also important to note. As an illustration, the advancement of multi-core processors and the growing accessibility of high-speed networks have made it possible to create new and more sophisticated real-time systems. These advancements have also brought about new difficulties, including the requirement for real-time operating systems to efficiently manage the resources of numerous cores and to offer real-time communication across various systems and devices in a networked environment \[34\].

2.4.1 Ubuntu Operating System

Ubuntu is a Linux-based operating system. It is developed for computers, smartphones, and network servers. This operating system is designed by a company called Canonical Ltd. The design of Ubuntu software is embedded in the principles of Open Source software development \[35\]. Ubuntu is composed mostly of free and open-source software. It is released in three various editions. Desktop, Server, and Core for Internet of Things devices and robots. All these editions can run on a computer alone, or in a virtual machine. Ubuntu is a famous operating system for cloud computing, with support for OpenStack. Ubuntu is released after every six months, and its long-term support (LTS) releases every two years \[36\].

2.5. Switched Ethernet

In real-time embedded systems Switched Ethernet is used to meet the requirements of bandwidth in automation, avionics, and the automotive industry. The IEEE 802.1 Time-Sensitive Networking (TSN) task Group \[4\] formed a set of standards such as time-triggered communication (802.1Qbv-2015 \[37\]), credit-based shaping (IEEE Std 802.1Qav-2009 \[38\]), frame preemption (802.1Qbu-2016 \[39\] and time synchronization (802.1AS2011) \[40\]. Previous Ethernet switches have had issues with timely and reliable delivery for real-time applications. To address this, new standards have been developed to support real-time applications over Ethernet networks. These standards rely on design-time synthesis and optimization to configure the network properly for real-time needs. Researchers have recently formulated new methods to optimize real-time application performance over Ethernet networks that meet these standards. \[41\].

2.6. Time-Sensitive Networking (TSN)

Time-sensitive networking (TSN) carries data traffic of time-critical applications over a switched Ethernet network shared by various kinds of applications having different requirements. TSN provides guaranteed data transport with low latency and worst-case end-to-end latency for time-critical traffic. TSN includes reliable time synchronization, a standard of IEEE 1588, which provides the basis for many other TSN features \[42\]. These features make TSN applicable to many use cases. TSN can be used in different domains, e.g., in industrial automation networks for Industry 4.0, in networks for critical machine-to-machine communication, networking in vehicles including support for autonomous driving, and many more. Standardization of this work is stated as IEEE 802.1. Audio Video Bridging (AVB) is successfully applied in different applications. AVB then evolved into IEEE 802.1. TSN has a broad scope of work. The standardization of TSN is a continuing effort in IEEE 802.1. New TSN features are being developed as the application of TSN is expanding, and further standardization is expected \[43\].
2.6.1 Time Synchronization

For time synchronization in the system, TSN uses the Precision Time Protocol (PTP) to synchronize the clocks to reference time. To highly precise network-wide synchronization in TSN, a capability of TSN was defined in IEEE 1588 [44], and it was extended in IEEE 802.1AS standard. The Time-Sensitive Networking (TSN) standards use the Precision Time Protocol (PTP) for time synchronization across the system. PTP allows clocks to be synchronized to a reference time [45]. To enable highly precise network-wide synchronization in TSN, the IEEE 1588 [44] standard defined capabilities for PTP. These capabilities were then extended in the IEEE 802.1AS standard to provide the level of synchronization needed in TSN networks. By leveraging PTP and enhancements from 1588 and 802.1AS [46], TSN can achieve the precise time synchronization required for real-time applications across the network.

For example, the standard specifies that end-to-end precision is below 1 microsecond over a 7-hops network. In the network, one node is assigned to be the Grand Master (GM) to play the role of a master clock. Other nodes play the role of slaves by synchronizing their clocks to the clock of GM. To estimate the clock offset between the clocks timestamps are used. The GM scatter Sync messages to advertise its clock. As the message takes some time to travel through the network, the slave requires network delay to calculate the offset to the master [47]. To achieve the requirements of real-time communication, the new protocol Medium Access Control (MAC) is executed on the wireless component [48] of the hybrid network with Ethernet TSN. Based on Time Division Multiple Access (TDMA), MAC introduces time synchronization means that are responsible for synchronizing various wireless nodes in the domain.

2.6.2 TSN Traffic Classes

This section explains the standards of the TSN and the classes of traffic in the TSN network. In the TSN network communication is carried out between end stations via links routes and switches through Ethernet communication.

- **ST traffic class**: A characteristic of ST traffic is that it is scheduled offline, thus making it fully deterministic having zero jitters during the delivery of the messages. A Time-Aware Gate (TAG) defined by the TSN standards [44] is controlled by the Gate Control List (GCL). This list defines the time slots of the network-wide reference time gates during which the gates must be open making the link available for the messages stored in the queues to be sent.

- **AVB Traffic Class**: The credit-based scheduling (CBS) defined by the AVB task group [44] specifies the credits for the AVB queues. The credit assigned to the messages in the queue is consumed once the message is sent over the network. In case there exists a pending message in the queue, the assigned credits either remain negative or are refreshed. For a message inside the AVB queue to be transmitted, its credits should either be positive or zero.

- **Best Effort (BE) traffic class**: The traffic belonging to this class has the lowest priority with no real-time assurances. The only condition under which it can send messages is that its gates are opened and that all the messages inside an AVB queue have been assigned a negative credit or the AVB traffic is not in a ready state for data transmission; as it is not controlled by the CBS.

2.6.3 Scheduling Mechanism

For the transmission of each frame, the TSN standard uses a queuing-forwarding mechanism, resulting in frames being stored inside queues on the output port of TSN-supported switches. The scheduler model in TSN describes whether the queue of frames is permitted to forward or not by enabling or disabling a transmission gate.

TSN has different scheduling mechanisms, briefly explain four mechanisms here [49, 50]:

- **CBS Mechanism**: CBS stands for a credit-based shaper. specifies the credits for the AVB queues. The credit assigned to the messages in the queue is consumed once the message is sent over the network.
• **FIFO Mechanism:** FIFO stands for first in first out. In this mechanism, frames are scheduled according to a first come first serve policy, which means that the scheduling is conducted in order of the arrival of frames whose gate is open for transmission.

• **SP Mechanism:** SP mechanism stands for strict priority mechanism. In this mechanism queues that have high priority will be scheduled first for transmission, then the low-priority queues will be transmitted.

• **Time-Aware Shaping (TAS):** The TAS scheduler transfer streams of traffic by opening and closing the gates. To assure low latency, this scheduler needs to be coordinated between network bridges. In the network, TAS needs synchronized clocks.
2.7. 4G-LTE

The Fourth Generation Long-Term Evolution, or 4G LTE, is a wireless technology that enables high-speed mobile broadband and internet access. The 3rd Generation Partnership Project (3GPP), a global collaboration of telecoms standards organizations, created and standardized it as a wireless communication standard. Since its first release in 2009, 4G LTE has emerged as the industry standard for mobile broadband and internet access, with billions of users worldwide relying on it for access to the internet on their mobile devices. It is an improvement over past generations of cellular technology, such as 2G and 3G, and it is intended to offer higher data speeds, lower latency, and more dependable connections [51]. With a 99.99% dependability rate, 4G can provide low latency in the region of 50 ms and high precision time synchronization in microseconds in favorable conditions [51]. The telecommunications sector has been greatly impacted by 4G LTE, which has also greatly accelerated innovation. It has facilitated the widespread use of mobile devices and given billions of people worldwide access to high-speed mobile internet. With its capacity to serve millions of linked devices through a single cellular network, 4G LTE has also been a crucial IoT enabler [51]. Also, By facilitating the delivery of high-quality video content to mobile devices, 4G LTE has also had an impact on other businesses, including the entertainment sector. Additionally, it has made it possible for brand-new software and services to be created, such as augmented reality and virtual reality, which have the potential to revolutionize how people use technology [51]. Additionally, With all the features needed for time-sensitive communications, 5G is viewed as a strong contender. However, the amount of work in the literature that exists in this field is quite small [51, 11]. Mäenpää and Sukuvaara in [52] compared 4G and 5G experimental network tests with IEEE 802.11 performance measurements. The study examined communication scenarios using IEEE 802.11 p-based 5G-4G networks. The 5G test network based on LTE Release 14, 4G (LTE), and IEEE 802.11 p communication was used for system-level performance evaluation of latency, data rate, and capacity from the infrastructure and end-user device viewpoints. According to their analysis, the most important vehicular application characteristics, including mobility, delay, bandwidth, coverage, and baud rate, can be better met by cutting-edge 5G radio interfaces.

2.8. Hardware Considerations

Due to its numerous benefits over conventional networking options, hardware platforms that mix these two technologies (Wired TSN and Wireless networks) have seen an increase in demand in recent years. For instance, TSN offers deterministic, low-latency communication, which is necessary for many onboard real-time applications. In contrast, 4G offers high-speed internet connectivity over many remote applications, which is necessary for many IoT and machine-to-machine (M2M) applications [53]. For a variety of applications, a hardware platform with both TSN and 4G capabilities provides a flexible and potent option. While 4G is a mobile communication technology that offers high-speed internet connectivity, TSN is a set of protocols and technologies that enable real-time communication over Ethernet networks. Also, a wide range of applications, including industrial control systems, robotics, and autonomous vehicles, are increasingly utilizing single-board computer (SBC) hardware platforms with TSN and 4G capabilities. Because they are frequently lightweight and affordable, these platforms are usable by a variety of users, from students and hobbyists to business professionals. Additionally, by combining TSN and 4G capabilities on a single platform, separate networking components are not required, which lowers the system’s overall cost and complexity. As a result of this connection, the system is also more flexible and scalable and can be quickly expanded to meet changing needs [53]. The CPU, which is in charge of carrying out calculations and carrying out instruction execution, is one of the crucial parts of an SBC with TSN and 4G capabilities. The memory, input/output ports, and peripheral components are frequently integrated into the microprocessor. The memory, which can be either volatile (RAM) or non-volatile (ROM) memory, is used to store data and program instructions. The SBC can communicate with other devices and systems, such as screens, keyboards, or network devices, thanks to input/output interfaces.
2.9. Integration of Wired and Wireless Network

Modern telecommunications and computing face a significant problem with the convergence of wired and wireless networks. The requirement for seamless and secure communication between wired and wireless networks has grown in significance as the number of connected devices increases. Increased flexibility, better scalability, and improved security through traffic segregation are just a few advantages that come with combining these two kinds of networks. High-speed and dependable communication for a variety of devices and applications is offered by wired networks like Ethernet. They are frequently used to link desktop PCs, servers, and other devices in a single physical place and are particularly suited for local area networks (LANs). However, the drawbacks of wired networks, such as the requirement for physical cables and the challenge of adding or moving devices, make them less appropriate for mobile devices and other applications that demand mobility and flexibility of movement. Although wireless networks, including Wi-Fi and cellular networks, provide more mobility and flexibility of movement, they also have some drawbacks, such as slower speeds and reliability as well as greater sensitivity to interference and security risks. When wired and wireless networks are combined, users get the best of both worlds: high-speed, dependable connectivity for devices that need it, as well as the mobility and flexibility of movement other devices and apps need [54]. Wired and wireless networks can be combined in a number of ways, such as by employing wireless access points (WAPs) to link wireless devices to a wired network, wireless bridges to link two wired networks or wireless mesh networks to build expansive wireless networks. Each of these strategies has pros and cons, and the ideal strategy will depend on the particular demands of a particular network [55].
3. Related Work

The state of the art in 4G LTE, wireless TSN, and time synchronization for converged wired-wireless networks is presented in this section.

3.1. Wired TSN

Mubeen et al. [56, 57] presents the modeling of TSN communication in distributed embedded systems. The approach consists of needful elements in a sequence and information on a TSN network particularly, which any component model should include to support TSN. The Network Specification (NS) element incorporates point-to-point networks like Switched Ethernet, AVB, and TSN. The NS element has properties of protocol, configuration, and details of timing. For communication of point-to-point networks, the model of the network includes one or more switches. The Rubus Component Model (RCM) includes the properties of the TSN switch [56]. This switch model controls a set of ports and links for communication between nodes and switches. The method of developing the model having TSN properties in RCM provides support for this thesis to achieve low latency [56]. TSN has a set of sub-standards specified by the IEEE 802.1 TSN Task Group to support communication on standard Ethernet. For reliable packet delivery with low latency in real-time applications, TSN depends on a central management solution that uses time scheduling. Capabilities of TSN can be mapped from Ethernet onto wifi, without architectural changes or protocol translation gateways. Authors in [58] use TSN features in wifi 7 development to achieve low latency. All those features of TSN will also help in this thesis to achieve low latency while sending messages in real-time between the nodes of the system. In [59], authors explain IEEE Std 802.1 and IEEE Std 802.3 as the standards for Time Sensitive Networks. These networks are comprised of featured bridges that are connected with each other through Ethernet links with standard MAC/PHY layers. Real-time applications require zero loss of packet because of buffer congestion, extremely low loss of packet because of failure of equipment, and guaranteed upper bounds on end-to-end latency. Explained standards of time-sensitive networks have been added to the wired TSN. These standards improve the performance of wired TSN and they will provide support in this thesis to overcome the issues in real-time communication when TSN will be integrated. The contribution of this thesis is to develop a communication network with the capabilities of predictable low latencies in the domain of real-time distributed embedded systems.

Linking the Ethernet TSN with the wireless network, clock synchronization is one of the challenges that need to be considered. Accurate time synchronization is required for hoping wireless networks to operate in a highly geared fashion with Ethernet TSN credentials. PTP and FTM are time synchronization protocols for communication that are explained in [60]. Researchers figure out that PTP is used both for time synchronization in Ethernet TSN and also on wireless whereas FTM is used for TSN time synchronization on wifi [61]. In this thesis, we will use PTP for time synchronization because of its high accuracy. This protocol can be implemented on TSN and 4G networks.

3.2. 4G LTE

A significant turning point in the development of wireless communication technology has been the creation of 4G LTE. Since its commercial launch in 2009, 4G LTE has supplanted other mobile broadband technologies as the industry standard, enabling billions of users worldwide to access high-speed data and voice services. Related research in the 4G LTE space has primarily addressed a number of important topics, including network architecture, radio resource management, quality of service, and security. Additionally, researchers have looked into a number of techniques for improving the functionality of 4G LTE networks, including dynamic spectrum management, interference reduction, and energy efficiency. A watershed monitoring system using a roboboat and 4G is implemented by [55] and a 4G network was used for data exchange between the roboboat and an android phone. The data transfer speeds for transmitting commands from the app on an Android smartphone to control a roboboat in outside conditions achieved an average speed of 1.84 seconds, and the time for displaying photographs on the app was 5.68 seconds with an image file size of about 4MB [55]. This Research Resulted in a large latency value and Real-Time Operating
systems and TSN were not utilized in the System.

Zungo et al [62] presents an innovative human-assisted framework for quay wall and ship hull inspection that uses Autonomous Surface Vessels (ASV) carried Remote Operated Vehicles (ROV) fitted with a variety of cameras and sensors to increase automation and accuracy. They concentrate on the design of the communication interface between the shore control station and the ROV in their work, which must be carefully considered to ensure optimal system operations. They rely on a cellular-based communication architecture, and also test the effectiveness of the system using various configurations while taking both 4G and 5G deployments into consideration. The results show that cellular technologies are able to meet the Quality of Service (QoS) requirements envisioned for the inspection services and therefore can be used to support unmanned vessels during their operations.

Sato et al [63] suggested a system for remote driving of automated vehicles combining 5G and 4G networks with an in-vehicle camera video transmission system. In the suggested system, a 4G network was utilized to reliably transmit low-bit rate videos while 5G networks are used to send high-quality and low latency videos. The in-vehicle camera video via the 5G networks is presented on the remote operator console when there are no interruptions to the 5G network’s in-vehicle video feed. When it is disrupted, the display is switched to the video via the 4G network, which acts as a backup network to ensure the dependability of remote driving even when the 5G network experiences an unexpected decrease in network quality.

A 4G-Connected Micro-Rover With Infinite Range was implemented by [64] using Mavlink. Hence, according to the research, the benefit of employing a 4G-LTE network for command and control over a unique network built specifically for this use case is as follows. First and foremost, there is already a network infrastructure that can be utilized very cost-effectively. Additionally, the driver and the vehicle can be separated by any distance thanks to the network infrastructure. The network architecture can accommodate millions of users, in the end. As a result, this strategy is better than creating and maintaining a system just for remote vehicle control. Furthermore, To coordinate the Mavlink packets from the car to the driver, a cloud-based IP address Linux instance was employed. The research also claimed that any Linux cloud machine or a basic virtual network like ZeroTier will be sufficient in delivering a static IP address that will help in interacting with the rover via 4G LTE. However, the research employed the cloud provider AWS.

A time-sensitive network examination of various broadband technologies, including WI-Fi, 4G, and 5G, revealed that 5G is the only wireless technology whose specifications are likely to include all the capabilities required for time-sensitive communications. Mission-critical IoT, industrial automation, and other deterministic application scenarios are best suited for 5G and even Wi-Fi 7 (802.11be) will not be able to ensure entirely predictable communications [51]. Nevertheless, many of the drawbacks of current Wi-Fi can be overcome in upcoming generations to make it ideal for time-sensitive networking [51].

### 3.3. Integration of TSN with Predictable Applications

A recent development in wireless communication called Wireless Time-Sensitive Networking (WTSN) promises to enable the predictable and deterministic transfer of time-critical data. Real-time control and device coordination are vital in a number of applications, including industrial automation, autonomous driving, and aerospace. Few studies have been done on WTSN in recent years to solve a variety of issues, including guaranteeing end-to-end latency and jitter boundaries, synchronizing clocks between devices, and facilitating multi-hop communication. The creation of the IEEE 802.1 TSN standard, which specifies a collection of protocols and techniques for deterministic communication in Ethernet networks, is one of the major contributions to the field of WTSN. The standard addresses topics like frame preemption, traffic scheduling, and time synchronization, which are essential for ensuring a predictable and consistent transmission of time-critical data [16].

While wireless network provides communication across the network’s edge, extending the borders where wired TSN is not practical owing to costs or a lack of mobility, wired TSN is anticipated to be the network architecture that will handle the majority of the traffic [54]. Seijo et al [54] researched the issues in the integration of hybrid networks, such as clock synchronization, and constructed a hybrid model as a demonstration of an idea by connecting wired TSN and Wireless Local Area Networks (WLAN) using a domain translator. Three potential integration models were
Analysed: 1. Wireless TSN that is non-TSN synchronized and priority Base, 2. Wireless TSN that is time-aware but non-TSN synchronized 3. TSN synchronized and time-aware-based Wireless TSN were all analyzed to find their latency performance and ultimately, this hybrid network delivered assured latencies of 150 µs with 3 TSN hop.

Two TSN-5G hybrid network topologies were implemented by [65], one was based on Ethernet TSN 802.11 and the other on Ethernet TSN-SwHARP, both include end-to-end clock synchronization capabilities. These topologies were able to estimate the achievable clock synchronization in each structure. The tests revealed that the designed architectures could effectively synchronize clocks to meet TSN criteria. Rost et al in [66] proposed that a 5G Bridge is capable of sustaining an expected latency of 1ms and below even for 15 kHz subcarrier spacing, which is the subcarrier spacing already employed in 3GPP 4G, according to the study on the performance of integrated 3rd Generation Partnership Project (3GPP) 5G and TSN Network. Additionally, a 5G Bridge can guarantee service even when numerous apps are running on the same mobile network. It was found that the bridge between the technologies is a key component of this integration.

Kehl et al [14] Utilized an Intel i210 single-board computer to simulate a factory cloud and a 5G Integrated with TSN Prototype for Edge-Controlled Mobile Robot. The operating systems’ kernel was partially preemptive thanks to the Linux Preempt RT Patch. The prototype consists of a 5G URLLC test system and commercially available TSN switches. Different TSN protocols were integrated with the 5G system for deterministic over-the-air communication. The paper describes the architectural details and step-by-step analysis of jitter reduction mechanisms applied in the communication chain between the application hosted in a factory cloud and the mobile robot. According to the study, time synchronization over the 5G system can be accomplished with high precision and the end-to-end latency over the TSN network and the 5G system was below 0.8ms with a 99.9% reliability, and the end-to-end time error introduced by synchronizing over the TSN network and including the 5G system was observed to be lower than 8s with a mean value of below 3s in the experimental validation over the test period of more than 6hrs. The peculiarities of the 5G architecture and the wireless characteristics, however, led to a jitter in the transmission across the 5G system in the order of 500s. Consequently, it was determined that 5G, especially when connected with an industrial TSN is capable of supporting demanding industrial automation use cases [14].

Gundallet al[67] Integrated 5G with TSN as a Prerequisite for a Highly Flexible Future Industrial Automation with Time Synchronization based on IEEE 802.1AS. To validate their result, three kinds of measurement were carried out (1) synchronizing two mini PCs with the standard IEEE 802.1AS (gPTP) synchronization mechanism without any wireless communication, (2) tunneling the IEEE 1588 (PTP) messages on UDP basis over the 4G system, and (3) applying a new concept for the integration of 3GPP 5G with IEEE 802.1AS.

Barhia and Saud [11] experimented to integrate TSN with 4G and compute the performance of the TSN-4G network by measuring the communication latency over the network plus end-to-end latency based on various sources of interference. The results showed that the latency for TSN-4G is 5789 ms, whereas that of TSN-WiFi is 398 ms. In their research, There was no real-time operating system employed in the experiment.

This thesis utilizes the Linux RT patch Real-time Operating System to provide the necessary abstraction layer, which ensures predictable and reliable latency. This is in contrast to previous research that did not employ a real-time operating system. Additionally, the architectural system employed in this thesis uses three nodes, as opposed to just two nodes in previous studies [11]. In this thesis, the first node is connected to the second node through a wired TSN switch, and the second node connects to the third node through a 4G LTE wireless medium while in the previous study, the first node is connected directly to the second node through 4G LTE [11]. It is also worth noting that this thesis utilizes single-board computers as nodes, rather than personal computers with a Windows operating system, as the nodes in the architectural system as implemented in the previous research [11].

Previous research has focused on various aspects of TSN-5G networks, including network architecture, protocol design, and performance evaluation. For example, few studies have explored the use of TSN protocols in 5G LTE networks, and have shown that TSN can significantly improve the real-time performance of LTE networks for delay-sensitive applications and little or no research
has focused on 4G LTE using real-time operating systems.
4. Research Method

4.1. System Development Research Method

The Research Method for System Development presented in Figure 1 is used for this thesis as it expedites research where system development is a vital element of the study. As explained in the paper [68] it involves six phases.

- **Conceptualize Framework**: The phase discusses the construction of a conceptualized framework to be studied in this thesis. The research questions will be conferred in the context of the proposed conceptualized framework.

- **Study State-of-the-Art**: An extensive literature review conducted for the study will be presented as a study of the state-of-the-art; based on which a research gap mapping onto the proposed conceptualized framework will be discussed.

- **System Architecture Development**: A detailed system architecture developed based on the proposed conceptualized framework will be presented and discussed.

- **System Modelling and Analysis**: The phase consists of further two steps:
  - Based on the conceptualized framework, extracted information from the literature review, and the constructed system architecture, a model depicting the system to be developed and analyzed will be modeled.
  - The developed system model will be analyzed for its correctness before moving toward the next phase of our research methodology.

- **System Implementation**: This phase of the research methodology adopted includes implementing the analyzed system model along with the integration of the hardware components involved. The code generated during this phase will then be transferred onto integrated hardware.

![Figure 1: Adopted Research Methodology](image)
• **Experimental Analysis:** A detailed experimental design will be proposed for evaluating the developed system. This phase includes the process of empirically evaluating the proposed system based on the defined experimental design. Further on, a separate section will discuss the results obtained along with its analysis.

### 4.2. Platform-Based Development

Similar to the previously mentioned methodology, Platform-based development (PBD) which is associated with embedded system development focuses on creating modular systems that can be easily adapted and extended. This approach is often used in the development of software and hardware systems, where the goal is to create a core set of components that can be reused and extended to meet the needs of different applications. In platform-based design, the focus is on creating a flexible and scalable architecture that allows for the easy integration of new components and functionality. This approach is well suited to complex systems where there is a need to balance the trade-off between generality and specificity, as it allows for the creation of a common platform that can be customized for specific use cases. In the context of a thesis, platform-based design can be a helpful approach when evaluating and improving existing systems [69].

### 4.3. Discussion

For the purpose of this thesis, we will opt for system development over platform-based development as the research approach in order to address the posed research problems. The main distinction between these two techniques is that PBD starts with an abstracted mode [70] while system development takes the approach of building a system over the course of the process while refining theories[70]. Since this thesis is not extending an already existing prototype and also not only designing a prototype but is going to implement the prototype through a research process, Hence, the System Development Method is a suitable research methodology for this thesis because it allows one to continually refine and change the direction of the thesis during every step based on the results of the preceding steps. In this thesis, we’ll create a method for prototyping the TSN-4G network and measure the end-to-end latency using an experimental analysis. Beginning with the state-of-the-art, the evaluated strategy in this thesis looks at how TSN integrates with 4G. The next step is to create the TSN-4G architecture, and a general prototype or model will be created on which experiments will be performed to assess the thesis’s objective.
5. Prototype Design

This section describes the prototype utilized for modeling the integration of Time-Sensitive Networking (TSN) and Fourth-Generation (4G) technology. This thesis uses the terms Single board computer (SBC) and node interchangeably. Since the primary focus of this work is to calculate the end-to-end latency, hence, three types of latency are discussed: communication latency, processing latency, and end-to-end latency. Communication latency refers to the time it takes for data to be transmitted from one node to another on a particular network and processing latency refers to the time it requires a specific node to process the Data. In contrast, end-to-end latency comprises both the processing or computation time that a node takes during task execution and the communication latency on the network. This section presents a comprehensive discussion of the different scenarios considered and is accompanied with block diagrams that provide an overview of their functioning. The following subsections will present the pseudocode for the sender and receiver nodes.

The prototype involves three nodes: Node A serves as the sender, while Nodes B and C act as receivers. Task scheduling functions are implemented to execute at specific time intervals, with tasks being periodic in nature. Node A features a “Data task” that is responsible for generating the frame and is prioritized with the highest possible priority. Additionally, it includes a “SendData task” that acts as a client and is tasked with transmitting data to Node B, logging processing latency, and maintaining second highest priority.

Node B encompasses a “ReceiveData task” which acts as a server and is assigned the highest priority for receiving frames from Node A over TSN. It also logs communication latency over TSN. Additionally, a “SendData task” which acts as a client is prioritized second-highest, responsible for forwarding frames to Node C over the 4G network. This task also logs node processing latencies.

On the other hand, Node C’s “ReceiveData task” which acts as a server, has the highest possible priority for receiving frames over 4G, and calculating communication latencies. Additionally, it includes a “ProcessData task” responsible for displaying frames and calculating processing latencies.

Node A’s functionality can switch between sender and receiver depending on its task scheduling. The “SendData task” runs every 10 unblocking milliseconds, while the “ReceiveData task” operates every 20 unblocking milliseconds. All nodes in this prototype utilize a standard Linux kernel and a Linux real-time patch operating system. Detailed information on the operating system configurations of each node will be provided in the experimental evaluation section.

5.1. Communication and Processing Latency over TSN-4G

To describe the scenario depicted in Figure 2, there are three Nodes involved: Node A, Node B, and Node C. Node A and Node B are connected to each other using a wired Time-Sensitive Network Switch, while Node B and Node C are connected through a 4G Network connection.
Figure 2: Communication Latency over TSN-4G

5.1.1 Block Diagram

The operational methodology of the three Nodes involved in the aforementioned scenario is illustrated in Figure 3. From the diagram, Node A which acts as the client is responsible for capturing, processing, and transmitting the intended video frame. On the other hand, Node B serves as both the Server and Client. In its capacity as the receiver or server, Node B accepts the data frame from Node A and assesses the communication latency over the TSN network. Additionally, Node B, as the sender node or the client, executes some internal tasks. Firstly, it converts the video frame format from the TSN format to the 4G protocol, following which it transmits it to Node C using the 4G network. The end-to-end delay duration, i.e., the time elapsed from Node A to Node C, is computed by adding the processing time in Node A, the communication latency on the TSN switch, the processing time on Node B, the communication latency on the 4G network and the processing time on Node C.
Figure 3: Communication Latency over TSN-4G
5.1.2 Code Implementation

This subsection provides a pseudocode that explains how the tasks on Node A, Node B, and Node C are implemented. Socket programming with an Ethernet header is used to implement it. Socket programming is a computer networking technique that enables two processes or programs to interact with one another over a network. Within this programming paradigm, a socket denotes a software endpoint that establishes a connection between two or more devices on a network. Establishing a connection involves creating a socket on each device that is wishing to communicate. Subsequently, these sockets are bound to specific IP addresses and port numbers, where the IP address identifies the device on the network, while the port number identifies the specific system port being used by the application that is communicating through the socket. After creating and binding the sockets, the client program sends a connection request to the server program. The server program, in turn, listens for incoming connection requests and accepts the connection from the client. Once the connection is established, the client and the server can exchange data through the socket by sending and receiving data.

Communication between the client and the server can be either connection-oriented or connection less. In connection-oriented communication, a persistent connection is established between the client and the server throughout the communication’s entire duration. On the contrary, in connectionless communication, each message is sent independently without establishing a persistent connection between the client and the server. Thus, communication latency is calculated using the timestamps of both the sender and the receiver [71]. Hence, from the pseudocode, the client code on Node A with algorithm 1 begins by initializing and configuring a socket for TSN communication to transmit video frames. It involves setting up a network socket, configuring the MAC address, and establishing the Ethernet header. The use of “hton” (host to network) conversion is crucial in networking programming to ensure that data is correctly translated for transmission over the network, converting it from the byte order of the host to the network byte order (typically big-endian) [72]. Subsequently, the video frames are captured, encoded into JPEG format, and then transmitted to Node B, and finally, the latencies are saved. This sequence of steps ensures efficient and reliable communication of video data over the network.

Algorithm 1 Video Transmission On NODE A

1. Initialize:
   2:   sockfd ← CreateSocket()
   3:   sa ← ConfigureSocketAddress("enp0s31f6")
   4:   eth_header ← ConfigureEthernetHeader()
   5:   cap ← OpenVideoCapture(0)
   6:   latencyFile ← OpenLatencyFile("./latency.txt")
2. Main Loop:
   8:   while 1 do
    9:     start_time ← GetCurrentTimeMillis()
   10:    videoFrame ← CaptureVideoFrame(cap)
   11:    buffer ← EncodeVideoFrame(videoFrame)
   12:    UpdateEthernetFrameSize(frame, eth_header, buffer)
   13:    SendFrameOverEthernet(sockfd, frame, sa)
   14:    end_time ← GetCurrentTimeMillis()
   15:    latency ← end_time − start_time
   16:    WriteLatencyToFile(latencyFile, latency)
   17:    SleepForOneSecond()
18:   end while
19. Cleanup:
   20: CloseLatencyFile(latencyFile)
   21: CloseSocket(sockfd)
   22: ReleaseVideoCapture(cap)

Node B is equipped with both client and server functionalities which make up the receivedata task and the sendData task. The server component is responsible for accepting frames from Node
A, while the client module manages the forwarding of these frames to Node C. Within Algorithm
2, the operation involves receiving frames from Node A, evaluating the communication latency
over the TSN connection between Node A and Node B, and then converting the byte order of the
frame from network byte order to the expected byte order of the host system using ntohs [72].

The function ntohs, standing for “network to host short”, acts as the complement to htons.
While htons converts a 16-bit unsigned integer from the host byte order to the network byte order,
ntohs reverses this operation by converting a 16-bit unsigned integer from the network byte order
to the host byte order. This conversion is vital for ensuring accurate interpretation and utilization
of the received data by the program [72]. Furthermore, the communication latencies are logged
for subsequent analysis. Additionally, to ensure data integrity and consistency, the reception of
frames is synchronized with a mutex, thereby preventing access to the received frame until all 1500
bytes have been successfully received. This approach safeguards against potential data corruption
or inconsistencies that may arise from concurrent access to incomplete frame data.

Following this, the client code (processData Task) on Node B takes on the task of sending the
data to Node C. Before initiating transmission, the frame is unlocked using a mutex, ensuring that
it can be accessed for further processing. Subsequently, the frame undergoes conversion from the
system frame format to a format suitable for transmission over the 4G network using htons (host
to network). This conversion step is crucial for ensuring that the data is properly formatted and
compatible with the 4G network’s transmission protocols.

Moreover, after the conversion process, the processing latencies are recorded for analysis pur-
poses.

Upon successful transmission, the data is routed to Node C, where it is received by the server
code. It’s important to acknowledge that the implementation of these codes may differ between
each node, reflecting their unique requirements and configuration.

Node C operates exclusively with server code, described in Algorithm 3. It contains a “Receive-
Data task” tasked with receiving data from Node B and computing latencies over the 4G network.
This task also logs the communication latencies to facilitate performance analysis.

Moreover, the “ProcessData task” undertakes the responsibility of converting network byte
order to host byte order using ntohs, decoding the frame, and ultimately displaying it. This task
is crucial to ensuring that the received data are correctly interpreted and displayed.
Algorithm 2 Video Reception and Transmission On NODE B

1: **Initialize:**
2:   sockfd\_receive ← CREATE\_RAW\_SOCKET()
3:   sa ← CONFIGURE\_SOCKET\_ADDRESS("enp0s31f6")
4:   frame\_mutex ← INITIALIZE\__MUTEX()
5:   communication\_latency\_file ← OPEN\_FILE("communication\_latency\_intermediate.txt")
6: **Main Loop:**
7:   while true do
8:     Communication\_start\_time ← GET\_CURRENT\_TIME\_MILLIS()
9:     frame ← RECEIVE\_FRAME(sockfd\_receive)
10:    Communication\_end\_time ← GET\_CURRENT\_TIME\_MILLIS()
11:    Communication\_latency ← end\_time − start\_time
12:    WRITE\_TO\_FILE(communication\_latency\_file, latency)
13:    LOCK\__MUTEX(frame\_mutex)
14:    UPDATE\_SHARED\_DATA(received\_frame, DECODE\_VIDEO\_FRAME(frame))
15:    UNLOCK\__MUTEX(frame\_mutex)
16:    processing\_start\_time ← GET\_CURRENT\_TIME\_MILLIS()
17:    LOCK\__MUTEX(frame\_mutex)
18:    frame\_to\_send ← CLONE(received\_frame)
19:    UNLOCK\__MUTEX(frame\_mutex)
20:    frame\_data ← ENCODE\_VIDEO\_FRAME(frame\_to\_send)
21:    SEND\_FRAME\_OVER\_UDP(send\_sockfd, frame\_data)
22:    processing\_end\_time ← GET\_CURRENT\_TIME\_MILLIS()
23:    processing\_latency ← processing\_end\_time − processing\_start\_time
24:    WRITE\_TO\_FILE(communication\_latency\_file, processing\_latency)
25:    SLEEP\_FOR\_FIVE\_SECOND()
26: end while
27: **Cleanup:**
28:   CLOSE\_FILE(communication\_latency\_file)
29:   CLOSE\_SOCKET(sockfd\_receive)
30:   RELEASE\_VIDEO\_CAPTURE(cap)
Algorithm 3 Video Reception and Processing On NODE C

1: Initialize:
2:   sockfd ← CREATEUDPSOCKET()
3:   addr ← CONFIGURESOCKETADDRESS("192.168.191.234", 100)
4:   frame_mutex ← INITIALIZEMUTEX()
5:   processing_latency_file ← OPENFILE("processing_latency.txt")
6:   communication_latency_file ← OPENFILE("communication_latency.txt")
7: Main Loop:
8:   while true do
9:      Communication_start_time ← GETCURRTIMEMILLIS()
10:     frame ← RECEIVEFRAME(sockfd)
11:     communication_end_time ← GETCURRTIMEMILLIS()
12:    communication_latency ← communication_end_time − communication_start_time
13:    WRITEToFile(communication_latency_file, communication_latency)
14:   LOCKMUTEX(frame_mutex)
15:   received_frame ← DECODEVIDEOFRAME(frame)
16:   UNLOCKMUTEX(frame_mutex)
17:   processing_start_time ← GETCURRTIMEMILLIS()
18:    DISPLAYRECEIVEDFRAME(received_frame)
19:   processing_end_time ← GETCURRTIMEMILLIS()
20:   processing_latency ← processing_end_time − processing_start_time
21:   WRITEToFile(processing_latency_file, processing_latency)
22: end while
23: Cleanup:
24:   CLOSEFILE(processing_latency_file)
25:   CLOSEFILE(communication_latency_file)
26:   CLOSESOCKET(sockfd)
27:   DESTROYMUTEX(frame_mutex)
5.2. Communication Latency over TSN

This section describes the high-level abstraction communication over the TSN. Node A, B, and TSN switch have unique IP addresses. Node A and Node B both have a TSN ethernet interface which is controlled by an ethernet controller that communicates with the TSN switch. The switch has four ethernet ports as shown in Figure 4. Any of these ports can be selected for communication between Nodes A and B. All the ports in the TSN switch can be configured through a web portal of the TSN switch.

![Figure 4: Communication Latency over TSN](image)

5.2.1 Block Diagram

This section describes the abstraction at the low level of TSN communication. Node A is the client side and has a Linux kernel or Linux real-time patch kernel operating system. Two tasks are defined in Node A, such as DataTask and SendData tasks. DataTask captures the video frames, and the SendDatatax sends those data over the ethernet interface to Node B via the TSN network. TSN network has a Time Aware Shaper (TAS) feature, which has its own Scheduler. This allows dedicated time slots for the transmission of data packets in real-time within the cycles. These cycles are divided into time slots as per TSN configuration and different time slots can be assigned to the different Ethernet ports. The Time Aware Scheduler picks the Ethernet frame to be sent not only following the strict prioritization plan at the queue, but also the circumstances of each queue. The scheduler decides which traffic queue can be transmitted at a specific point in time within the cycle. Node B acts as both server and client, but in this case of communication over the TSN, Node B acts as a server. This Node has three tasks as shown in Figure 5. DataTask and SendData, which are responsible for processing and sending the video frame over the 4G network interface to Node C when this Node acts as a client. The ReceiveData task is responsible for receiving the data from Node A.
5.3. Communication Latency over 4G

This setup involves a prototype comprising two nodes, namely Node B and Node C. The communication between these two nodes takes place using 4G technology. The primary aim of this setup is to evaluate the latency in communication over the 4G network when frames are sent from Node B to Node C as shown in figure 11.

In order for Node B and Node C to transmit or receive Data, they must complete specific tasks outlined in Figure 11 where socket communication is used to establish a 4G transmission by configuring the socket and server addresses, binding the socket to a port, receive data and close the file.

5.3.1 Block Diagram

Figure 7 illustrates the low-level responsibilities of the sender and receiver nodes over a 4G network. The sender node, identified as Node B (Client), is responsible for performing specific internal tasks, including configuring the socket, connecting to the server, and sending the data to the server in
Node C. On the other hand, the primary responsibility of the receiver node, which is Node C (Server), is to establish a connection with the client and receive the data. Once the 4G network processes the data, it is delivered to the receiver node (server). The duration of data transit time from the sender to the receiver via the network is determined by comparing the timestamps recorded by both the sender and the receiver.

Figure 7: Communication Latency over 4G
6. Experimental Evaluation

6.1. Experimental Setup

This section explains the experimental setup and how various hardware and software components are set up to conduct this experiment. Three single board computers (SBC) OK1028A-C act as three Nodes A, B, and C which are connected through TSN and 4G networks in a chain to perform this experiment. A TSN switch is connected between Node A and Node B via ethernet cable. A 4G module is installed in Node B to send and receive data from Node C which also has a 4G module and standard Linux or Linux real-time patch are employed in all the nodes.

6.1.1 Single Board Computer

![Figure 8: The OK1028A-C single-board computer](Image)

The experiment is carried out on the OK1028A-C single-board computer. The OK1028A-C is a development board and single board computer that utilizes a dual-core LS1028A processor up to 1.5GHz, with a design based on Cortex-A72. It includes 2GB of DDR4 RAM and 8GB of eMMC storage and has the capability to support 5 Gigabit Ethernet ports with TSN, as well as 2 CAN-FD, USB3.0, UART, SPI, IIC, LVDS, TF card slot, SATA3.0, and a headphone peripheral source. Additionally, it includes an M.2 Key B slot for a 5G module, a mini PCIe for a 4G module, and an M.2 Key E for a WiFi module [73].

6.1.2 Raspberry Pi Camera Module

The Raspberry Pi Camera Module is a compact and versatile camera add-on designed specifically for the Raspberry Pi single-board computer (SBC) and other single-board computers. It provides an inexpensive means of adding imaging capabilities to a project, from simple photo editing to complex computer vision applications. Installed in Node A, this camera records encoded video frames, which are subsequently sent to Node B via the TSN and, ultimately, Node C via the 4G network [74].
6.1.3 ZeroTier

ZeroTier is an SDN (software-defined networking) technology that makes it possible to create virtual networks with private and secure communication features. Its purpose is to facilitate smooth device communication over the Internet and streamline network configuration. It can be used for a number of purposes, such as establishing virtual LANs, connecting distant devices, and facilitating safe device-to-device communication via the Internet. It is a well-liked option for people and businesses searching for a safe, user-friendly Software Define Networking (SDN) solution because of its ease of use and adaptability [75]. Since the 4G network has a dynamic IP address, it is used to provide a static IP address between NODE B and NODE C over the 4G network.

6.1.4 TSN Switch

The TSN switch which is used in this thesis is the UG3506-MTSN kit. This TSN switch is a Multiport Time-Sensitive Networking (MTSN) switch and it supports TSN features such as Time Synchronization(IEEE 802.1AS), Time Shaping(IEEE 802.1Qbv), and Configuration of the network(IEEE 802.1Qcc). This TSN switch has 4 external ports (Port 0, Port 1, Port 2, Port 3) and 2 internal ports. SFP module is installed in the external ports which receive and send Ethernet signal between nodes. Port 0 and port 1 are used to connect Node A and Node B. This board can be configured by signing in to the TSN switch through its IP address.
6.1.5 4G Module

![EC25EUX-MINIPCIE](image)

The 4G module is EC25EUX-MINIPCIE which is a cellular module manufactured by Quectel Wireless Solutions. It is a Mini PCIe form factor module that supports 4G LTE networks and is designed for use in a wide range of applications, including industrial routers, industrial PDAs, and video surveillance systems. The module supports LTE Category 4, providing maximum download speeds of up to 150Mbps and maximum upload speeds of up to 50Mbps. It also supports multiple bands and provides global coverage. The EC25EUX-MINIPCIE module includes various features such as GNSS, voice, and SMS support, making it a versatile option for a range of applications that require cellular connectivity [76].

6.1.6 Ethernet Cable

CAT 6 ethernet cable is used to connect Node A and Node B. This ethernet cable supports the TSN switch.

6.1.7 Preempt Real-Time Patch

This thesis aims to achieve real-time data transmission between nodes and measure the latency rates both between nodes and end-to-end. Ubuntu was chosen as the operating system for this purpose, primarily because the single-board computer OK1028A-C utilized in this study supports Ubuntu. While Ubuntu is not inherently a real-time operating system, the implementation of the Preempt RT patch is crucial to transforming the system into a real-time environment. This patch enhances the Ubuntu kernel’s preemptiveness, allowing for real-time data transmission and ensuring low latency. The Preempt RT Linux scheduler is effective in guaranteeing constraints by preemption of threads, even those in critical sections such as the `send_data` and `receive_data` code sequences.

6.1.8 Traffic Characteristics

This section describes the characteristics of data sent from nodes A to C. A video frame of 1500 bytes is generated at node A and sent to node B and then node C. All the ethernet frames are processed and sent to the Switch’s output port. TAS scheduler sends the data from the output port to node B by checking the states of the gates and the time interval of the entity of the control list. All the data frames of some bytes are scheduled in the specific time slot with all active queues. Every queue has its own priority from zero(lowest) to seven(highest). The maximum cycle time to send data from node A to B is set in nanoseconds. The time interval of the time slot is defined in nanoseconds and it must not be greater than the maximum cycle time.
6.2. Experiments

This section delves into the design and execution of experiments aimed at assessing the end-to-end latency of the TSN-4G network. To carry out these experiments, specific prerequisites were essential, including a TSN switch, a Raspberry Pi camera, two 4G-enabled SIMs, two 4G modules, Ethernet cables, and three single-board computers equipped with Ubuntu operating systems. Initially, the data transmission concept was implemented independently over 4G and TSN networks. Subsequently, both networks were integrated into a unified entity, referred to as TSN-4G, and experiments were conducted to determine the end-to-end latency of the integrated TSN-4G network.

Throughout the experiments, video frames were initially transmitted using the standard Ubuntu operating system, followed by Ubuntu with a Real-time patch. The real-time path renders the kernel partially preemptible. Three distinct scenarios were devised in this project, each thoroughly discussed in this section. The first two scenarios involved sending data frames from one node to another over TSN and then over the 4G network, respectively. The third scenario encompassed the fully converged networks, integrating both TSN and 4G. In each scenario, video frames captured from a camera with identical specifications were sent to evaluate and compare the latency experienced by TSN and 4G under different operating system configurations – one with standard Ubuntu and another with a real-time patch.

To ensure precise latency measurements, Universal Coordinated Time (UTC) served as the global clock reference. The sender and receiver tasks (client and server) were implemented using the C programming language. Additionally, to facilitate uninterrupted traffic flow, firewall settings, and inbound/outbound traffic rules were disabled. A virtual private network, ZeroTier (VPN), was employed to maintain a consistent connection between the server and the client over 4G (Node B and Node C) since node B and node C exist in a different subnet mask, and this ensures a robust experimental setup.

6.2.1 Scenario 1: Communication between Node A and Node B by sending data frames over TSN

In this scenario, video frames using C programming are sent from Node A to Node B over the TSN network in real-time using the standard Ubuntu operating system and later using the Ubuntu with Preempt Real-Time Patch. At Node A two tasks have been created to generate data and send data to Node B. Data task is responsible for capturing video frames of 1500 bytes and the Send_data task is responsible for sending those frames to Node B. At Node B receive_data task receive those bytes. Software time stamping and Wireshark is used to measure the processing latency in the nodes and communication latency over the TSN network.

6.2.2 Scenario 2: Communication between node B and node C by sending data frames over 4G

In this scenario, data frames are being transferred via the 4G network from Node B to Node C, and all nodes are operating on the standard Ubuntu operating system and later using the Ubuntu with Preempt Real-Time Patch. To establish 4G connectivity, the EC25AFA-MINIPCIE is being utilized for the 4G communication. Socket programming has been implemented on both the sender and receiver nodes, and the 4G module facilitates the transmission of these video frames. ZeroTier is used to achieve a constant connection between node B and node C over a 4G network. Two independent tasks have been scheduled on each node to transmit and receive data at specific intervals. Timestamps have also been accurately recorded at both nodes to determine the time it takes for a frame to reach from one node to another.

6.2.3 Scenario 3: Communication between node A, node B, and node C over an integrated network of 4G and TSN

In this scenario, data frames are sent from Node A to Node B and then to Node C in real-time as shown in figure 3. Socket programming is implemented on all the nodes. Node A acts as a client because it sends video frames. Node B acts as a server and client at the same time because
it sends and receives frames and Node C acts as a server because it receives data only. Software timestamps have also been accurately involved in all nodes to determine the end-to-end latency.

6.3. Results

This section presents the results obtained from implementing the aforementioned scenarios.

6.3.1 Scenario 1

In this specific scenario, a 1500-byte encoded video frame, which is the maximum size for transmission over an Ethernet frame, is sent from node A to node B via the TSN switch, which operates at a speed of 100 Mbps. Both port 0 and port 1 of the TSN switch are activated to facilitate the data frame transmission, with port 0 serving as the sending port and port 1 as the receiving port. For the transfer of the 1500 bytes of data within each frame to the receiving port, two-time slots are designated, each assigned with active queues (Q0 and Q7).

To determine the transmission time of the TSN switch, the formula provided in Equation 6.3.1 is employed. This formula requires the bit count of the frame and the network speed as inputs. Upon applying the formula, a calculated value of 0.12144 ms is obtained. The following steps are employed to evaluate the transmission time of the frame on the TSN switch:

\[
\text{Frame Transmission} = \frac{\text{Number of Bits in a Frame}}{\text{Network Speed}}
\]

\[
= \frac{(\text{overhead + Frame Size}) * 8}{\text{Network Speed}}
\]

\[
= \frac{(18 + 1500) * 8}{100 * 10^6}
\]

\[
= 0.00012144 \text{ second}
\]

\[
= 0.12144 \text{ milliseconds}
\]

To assess the processing latency of the code execution on Node A, a set of a thousand frames, each comprising 1500 bytes along with corresponding timestamps, was utilized in the experiment. The outcomes of this experiment were initially obtained using standard Ubuntu and later with the addition of Ubuntu with RT_Patch, as illustrated in Figure 12. The results indicate that with Ubuntu and RT_Patch on Node A, the processing latency exhibits a maximum value of 29.01 ms, a minimum latency of 18.98 ms, and an average latency of 24.38 ms. In contrast, when Node A operates with standard Ubuntu, the processing latency records a minimum of 52.93 ms, a maximum of 67.03 ms, and an average of 59.84 ms.

The introduction of Ubuntu with RT_Patch renders the Ubuntu kernel partially preemptible, allowing the kernel to adhere to the priority utilization of the code. This accounts for the observed low latency in Node A when Ubuntu with RT_Patch is employed. Additionally, assigning the task the highest priority possible ensures its uninterrupted execution whenever it is ready, without being preempted by other tasks. Conversely, with standard Ubuntu on Node A and the same priority assigned, the processing task waits until all internal tasks with higher priorities have been completed, resulting in increased latency which shows that standard Ubuntu does not honor the priorities.

Furthermore, the processing and communication latencies within Node B were documented and visually represented in Figure 13, Figure 14, and Figure 15. It’s crucial to highlight that Node B’s communication latency specifically refers to the latency encountered via the TSN switch. When Node B operates with Ubuntu and RT_Patch, it exhibits a maximum processing latency of 13.07 ms, a minimum processing latency of 8.72 ms, and an average processing latency of 10.91 ms. Additionally, it demonstrates a maximum interference communication latency of 16.11 ms, a minimum interference communication latency of 10.06 ms, and an average interference communication latency of 13.08 ms. This latency is the accumulative latencies of the TSN frame transmission time and the external interference latencies over the TSN. The TSN frame transmission time is always deterministic with a rate of 0.12144 ms.
In the scenario where Node B runs with standard Ubuntu, it showcases a maximum processing latency of 19.83 ms, a minimum processing latency of 12.67 ms, and an average processing latency of 16.23 ms. Furthermore, it experiences a maximum interference communication latency of 28.1 ms, a minimum interference communication latency of 21.05 ms, and an average interference communication latency of 24.65 ms. Also, this latency is the accumulative latencies of the TSN frame transmission time and the external interference latencies over the TSN. The TSN frame transmission time is always deterministic with a rate of 0.12144 ms irrespective of the operating systems. These findings offer insights into the distinct processing and communication performance characteristics of Node B, emphasizing the impact of the operating system on these latencies and the deterministic nature of the TSN.

To assess the overall latency of node B relative to its operating systems, the summation of processing and communication latencies was conducted. Under the Ubuntu with RT_Patch configuration, node B registers a maximum total latency of 29.06 ms, a minimum total latency of 18.84 ms, and an average total latency of 23.99 ms. Furthermore, in the case of standard Ubuntu operation, node B exhibits a maximum total latency of 47.65 ms, a minimum total latency of 33.93 ms, and an average total latency of 40.89 ms. Notably, the latency values observed over the TSN remain relatively stable, independent of the operating system running on node B. This stability is attributed to the deterministic nature of the TSN, highlighting its consistent performance characteristics across different operating system configurations.
Figure 14: Node B TSN frame transmission latency and real-time external Interference Over TSN

Figure 15: Node B TSN frame transmission latency and Non-real-time external Interference Over TSN
6.3.2 Scenario 2

In this particular scenario, node B decodes the Ethernet frame received from node A, extracts the payload, converts it to the 4G network format for transmission to node C using the hton (“Host to Network”) API of socket programming, and then displays it on the screen. Since node B and node C are in different subnet masks, Zero-Tier is employed to establish a connection between them. Subsequently, the processing and communication latency of node C is evaluated, and the corresponding plot is displayed in figure 16 and figure 17.

![Figure 16: Processing Latency On Node C](image1)

![Figure 17: Communication Latency From Node B To Node C Via 4G](image2)

Furthermore, it is important to highlight that the communication latency of node C specifically refers to the latency experienced over the 4G network.

When operated with Ubuntu and RT_Patch, node C displays a maximum processing latency of 11.97 ms, a minimum processing latency of 6.73 ms, and an average processing latency of 9.33 ms. Additionally, it exhibits a maximum communication latency of 40.95 ms, a minimum communication latency of 31.99 ms, and an average latency of 36.59 ms.

Conversely, under the configuration of standard Ubuntu, node C showcases a maximum processing latency of 21.56 ms, a minimum processing latency of 16.51 ms, and an average processing latency of 19.02 ms. Furthermore, it illustrates a maximum communication latency of 82.82 ms, a minimum communication latency of 56.83 ms, and an average communication latency of 70.03 ms. These results underscore the distinct impact of operating systems on both processing and communication latencies of node C.
Integrating node C’s processing and communication latencies in relation to the operating system results in the node’s overall latency. When equipped with Ubuntu and RT_Patch, node C exhibits a maximum total latency of 52.63 ms, a minimum total latency of 39.16 ms, and an average total latency of 45.92 ms. In contrast, utilizing standard Ubuntu leads to node C having a maximum total latency of 104.03 ms, a minimum total latency of 73.91 ms, and an average total latency of 89.05 ms. Notably, irrespective of the operating systems, the 4G network introduces a slightly higher computation time. This is attributed to 4G’s relatively slower speeds, increased latency, congestion, weakened signals, and protocol overhead, all contributing to prolonged code execution times and data transfers between devices. To address these shortcomings, 5G is designed to provide quicker, lower latency, and more efficient connectivity.

6.3.3 Scenario 3

In this specific situation, we consolidate the overall latency, considering both processing and communication latencies, originating from nodes A, B, and C. The summation of these latencies is computed to determine the end-to-end latency based on the operating system utilized on each node. The outcomes of this analysis are illustrated in Figure 18 and Figure 19.

Consequently, when employing Ubuntu with RT_Patch as the operating system on the nodes, the frames encounter a maximum end-to-end latency of 106.76 ms, a minimum latency of 80.86 ms, and an average latency of 94.30 ms. In contrast, if the nodes exclusively use the standard Ubuntu operating system, the frames exhibit a higher maximum end-to-end latency of 215.32 ms, a minimum latency of 165.34 ms, and an average latency of 189.80 ms. This highlights the significance of the operating system choice in influencing latency performance, with Ubuntu with RT_Patch yielding more favorable results compared to the purely Ubuntu configuration.

Figure 18: Real-Time Average, Maximum, and Minimum end-to-end Latencies
6.4. Discussion

This section details the experiments conducted in the thesis, highlighting the challenges faced during system configuration. To establish unidirectional communications among all nodes, a ZeroTier VPN client was implemented between node B and node C. This setup facilitated connectivity not only among nodes within the local area network but also with those outside of it.

The primary objective centered around integrating two distinct networks and evaluating end-to-end latency, with a specific focus on comparing the impact of operating systems, namely standard Ubuntu and Ubuntu with RT_Patch. The RT_Patch, when installed in the Ubuntu kernel, introduces partial preemption. Initial experiments involved smaller-scale tests, including separate frame transmissions over the TSN switch and 4G module, followed by the calculation of processing, communication, and overall latencies. Subsequently, these individual scenarios were merged into a unified setup, and the processing, communication, and general latencies were computed.

Upon analyzing the integrated networks, it became evident that frames experienced significant interference during transmission from node A to node C within the TSN-4G network. Notably, an increase in latency was observed when the camera was exposed to bright light, regardless of the operating systems and compression formats used. This increase was attributed to the larger pixel count in the captured image, leading to longer processing times.

Figure 20 illustrates the overall average latency transmission of nodes with their respective operating systems. The legend distinguishes various latency components. From the legend, RT_PROC1 is the processing latency of node A with Ubuntu and RT_Patch, RT_PROC2 is the processing latency of node B with Ubuntu and RT_Patch, RT_COM2_TSN is the communication latency over TSN in node B with Ubuntu and RT_Patch, RT_PROC3 is the processing latency of node C with Ubuntu and RT_Patch, and RT_COM4_4G is the communication latency over 4G in node C with Ubuntu and RT_Patch. Consequently, NON_RT_PROC1 is the processing latency of node A with standard Ubuntu, NON_RT_PROC2 is the processing latency of node B with standard Ubuntu, NON_RT_COM2_TSN is the communication latency over TSN in node B with standard Ubuntu, NON_RT_PROC3 is the processing latency of node C with standard Ubuntu, and NON_RT_COM4_4G is the communication latency over 4G in node C with standard Ubuntu. Moreover, regardless of the operating systems in use, Node B consistently exhibits the lowest latencies. This can be attributed to the deterministic nature of the TSN. In contrast, the 4G network encounters considerable latency due to factors such as congestion, weakened signals, and protocol overhead. These elements collectively contribute to extended code execution times and slower data transfers between devices.
Upon comparing overall latencies, it is evident that when the operating system is real-time, the system exhibits an end-to-end maximum latency of 106.76 ms, a minimum end-to-end latency of 80.86 ms, and an average end-to-end latency of 94.30 ms. In contrast, with a non-real-time operating system, the system shows a higher maximum end-to-end latency of 215.32 ms, a minimum end-to-end latency of 165.34 ms, and an average end-to-end latency of 189.80 ms. This outcome suggests a promising reduction in latency when running with a real-time operating system compared to non-real-time counterparts. It implies that employing a fully preemptible operating system like freeRTOS or VxWorks could further drastically reduce latency.

![Figure 20: Overall Frame Transmission Characteristics](image)

Additionally, Integrating the Ethernet frame with socket programming, incorporating the User Datagram Protocol (UDP) over 4G, utilizing Linux (especially real-time Linux), and implementing a C program collectively contribute to the reduction of latency. Additionally, it’s crucial to emphasize that, when establishing a connection between two nodes through ZeroTier, it is imperative to allocate sufficient time for ZeroTier to stabilize before initiating data transmission.
7. Threats to Validity

During the course of conducting experiments, it is crucial to acknowledge that various factors can potentially impact the results. Identifying and addressing these potential threats to the validity of the thesis is essential for significantly improving the accuracy of latency rates. The following considerations highlight key aspects pertaining to potential threats to the validity of the obtained results.

7.1. Establishing Communication Channel

To create a communication channel between nodes, it is necessary to use6 -a private network mask or a static IP address, which can expedite the stabilization of connections among nodes with different subnet masks. In this thesis, the tool “ZeroTier” is employed to set the IP address as static.

7.2. Data Transmission

Among various methods available for data transmission between the nodes, the most efficient method was the implementation of socket programming using the available APIs of the operating systems rather than going for AT commands because 1. socket programming allows for direct communication between devices without the overhead associated with AT command processing [77]. 2. With socket programming, you have the flexibility to optimize communication protocols and network configurations for low latency. You can implement techniques such as using UDP (User Datagram Protocol) for real-time data transmission, reducing protocol overhead, and implementing efficient error-handling strategies [71]. 3. Socket programming supports concurrent communication, allowing multiple devices to send and receive data simultaneously. This concurrency can help reduce latency by enabling efficient use of network resources and minimizing wait times [71]. 4. For applications with real-time requirements, such as streaming media, gaming, or control systems, socket programming offers better support for meeting low-latency objectives compared to the relatively higher latencies associated with AT command processing [71].

7.3. Data Translation

Socket programming simplifies the translation of frames between Ethernet and 4G networks by providing functions like hton (host to network) and ntohs (network to host). These functions facilitate the conversion of data between the byte order used by the system and the byte order expected by the network. This capability streamlines the process of transmitting data across different networks, ensuring compatibility and efficiency. By leveraging these functions, developers can seamlessly handle the translation of frames, reducing complexity and potential errors in network communication.

7.4. Single Board Computer

Another factor posing a potential threat to the validity of our research is the reliance on a single-board computer equipped with TSN features that can be portable, as porting a real-time operating system into such a setup can be seamlessly implemented across various operating systems.

7.5. TSN Switch

The TSN kit utilized in this thesis is the outdated TSN switch. However, upgrading to the advanced TSN switch will reduce the latency rate because of its advanced software and hardware components.
8. Conclusion

This thesis set out to explore the integration of Time-Sensitive Networking (TSN) with 4G networks for real-time applications. The primary motivation was to harness the benefits of the deterministic, low-latency capabilities of TSN along with the ubiquitous connectivity and flexibility provided by 4G wireless technology. The key research questions addressed were: (1) What are the challenges in integrating TSN with a 4G network to fulfill the demanding requisites of real-time applications? and (2) How can TSN be effectively integrated with a 4G network to achieve predictable low latencies for real-time applications?

To investigate these questions, an experimental testbed was developed consisting of single-board computers acting as end nodes connected over TSN and 4G network segments. Three scenarios were evaluated: TSN-only, 4G-only, and converged TSN-4G communication. The end-to-end latency was measured under different operating system configurations and network settings.

Regarding the first research question, the key challenges identified were:

- Choice of the operating system significantly impacted latency, with the real-time operating system providing lower and more consistent latency compared to the non-real-time operating system.
- 4G networks experienced higher and unpredictable latency due to inherent factors like congestion, interference, and weakened signals.
- Synchronizing clocks between the TSN and 4G segments added complexity.
- Protocol translation between TSN and 4G data formats introduced overheads.

To address the second research question, effective techniques to achieve low, predictable latency over converged TSN-4G networks included:

- Using real-time Operating systems like freeRTOS or VxWorks for latency reduction and predictability.
- Configuring TSN features like time-aware shaper for scheduled, prioritized traffic.
- Direct interconnection between TSN and 4G to avoid protocol translation.
- Virtual private networks to ensure robust connectivity over 4G with devices in different subnet masks.

The results revealed that employing real-time operating systems like Ubuntu with RT Patch significantly reduced latency compared to standard Ubuntu, highlighting the importance of operating system choice. The TSN network exhibited consistently low latency unaffected by the operating system choice, due to its deterministic nature. However, the 4G network experienced higher, variable latency due to factors like congestion and interference. Synchronizing clocks between TSN and 4G using PTP was feasible but introduced some jitter. The converged TSN-4G network achieved reasonable end-to-end latencies on the order of 100ms when using real-time OS.

In conclusion, integrating TSN with 4G networks for real-time communication is promising but faces challenges. operating systems selection, synchronized clocks, schedule traffic, and avoiding unnecessary protocol translations help reconcile the deterministic TSN behavior with less predictable 4G performance. While 100ms end-to-end latency was achieved, further improvements are possible using 5G and a fully preemptive operating system. Overall, converged TSN-4G networks present a flexible approach to delivering robust, low-latency communication for real-time applications.

The findings provide insights into design decisions and tradeoffs involved in integrating wired TSN and wireless networks. Future investigations can build upon these results in areas like protocol optimizations, cross-layer scheduling, and hypervisor-based virtualization for converged TSN-4G systems. With enhancements and maturation of technologies like 5G and TSN, converged wired-wireless networks present an important platform for next-generation real-time applications across automation, transportation, and robotics domains.
9. Future Work

This thesis lays the groundwork for the integration of Time-Sensitive Networking (TSN) with 4G networks, showcasing its feasibility for real-time communication. However, several promising avenues exist for future research, offering opportunities to enhance and expand the current study:

- **TSN-5G Integration Performance Evaluation**: Building upon the achievements with TSN and 4G, future work could delve into the performance evaluation of TSN-5G integrated networks. The exploration of higher speeds and lower latencies inherent in 5G may uncover significant improvements, warranting a thorough analysis of their impact on end-to-end latency.

- **Implementation of Fully Preemptive Real-Time Operating Systems**: An additional avenue for future work involves exploring the use of fully preemptive real-time operating systems. Assessing their impact on latency and determinism within the context of integrated TSN-5G networks could provide valuable insights into achieving more stringent real-time requirements.

- **Quality of Service (QoS) Mechanisms**: Investigating mechanisms to provide Quality of Service (QoS) guarantees for different traffic types within the converged network is a crucial next step. This would enable the support of diverse application classes, contributing to the adaptability and versatility of the integrated TSN-4G system.

- **Incorporating Fog/Edge Computing**: The integration of fog and edge computing technologies represents an intriguing avenue. Future research could explore the implementation of real-time data processing and control at the network edge. Evaluating potential latency improvements resulting from edge computing introduces a challenging yet rewarding dimension to the study.

- **Development of Industry-Specific Architectures**: Tailoring architectures and protocols specifically for Industrial Internet of Things (IIoT) applications over converged TSN-cellular networks is another area ripe for exploration. Shifting away from reliance on legacy protocols has the potential to significantly enhance efficiency in industrial settings, a domain critical for real-time applications.

- **Utilizing Virtualization for Flexibility**: The integration of virtualization through hypervisors and containers can offer increased flexibility and resource management within integrated TSN-4G networks. Future work could explore how these virtualization technologies contribute to scalability and adaptability, ensuring the network’s robustness in dynamic scenarios.

Incorporating these proposed directions into future research efforts would not only deepen the understanding of integrated wired TSN and wireless networking but also contribute valuable insights to the broader field of real-time communication. Addressing the outlined aspects within future work provides a comprehensive roadmap for advancing the current study and offers a robust foundation for ongoing research endeavors in this crucial domain.
10. Acknowledgments

We extend our deepest gratitude to our university supervisors, Saad Mubeen and Zenepe Satka, for their invaluable guidance, support, and mentorship throughout the entire duration of our thesis. Their expertise, insightful feedback, and unwavering encouragement have played a pivotal role in shaping our research and enriching our academic journey.

We also express sincere appreciation to our supervisor, John Lundback, at Arcticus System in Jarfalla, Sweden. Their practical knowledge, industry insights, and consistent encouragement have been instrumental in bridging the gap between theory and practice. Their guidance significantly contributed to the practical relevance of our research.

We are thankful to Malardalens University for providing a conducive research environment and access to necessary resources. Their commitment to academic excellence has served as a constant inspiration throughout our studies.

Special thanks to the Swedish Institute for awarding the scholarship that enabled Ekpenyong Esu Ekpenyong to pursue studies in Intelligent Embedded Systems at Malardalen University, Vasteras.

Lastly, we express gratitude to our friends and family for their unwavering support and understanding during this challenging endeavor. Their encouragement, patience, and belief in our abilities have been a constant source of motivation. Their contributions have been invaluable in shaping the outcomes of this study.
References


A Appendix 1

This C code includes two distinct tasks: the "Capture Task," responsible for capturing frames from the camera, and the "SendData Task," which transmits these frames over TSN. These tasks are scheduled to run at specific intervals. The implementation involves client interactions utilizing Ethernet headers and UDP for communication. To ensure data integrity, a semaphore is utilized to protect critical sections. Priority assignment is established, with the "Capture Task" given the highest priority, while the "SendData Task" is assigned a lower priority. The code is designed to facilitate the transfer of captured frames from NODE A to NODE B through the TSN switch.

```
#include <arpa/inet.h>
#include <linux/if_packet.h>
#include <net/if.h>
#include <net/ether.h>
#include <net/inet.h>
#include <net/udp.h>
#include <opencv2/opencv.hpp>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <unistd.h>

#define FRAME_SIZE 1500

// Global variables
cv::Mat videoFrame;
pthread_mutex_t frameMutex = PTHREAD_MUTEX_INITIALIZER;

struct ether_header eth_header;
struct iphdr ip_header;
struct udphdr udp_header;
char frame[FRAME_SIZE];

int sockfd; // Declare sockfd globally
FILE *latencyFile; // Declare latencyFile globally
FILE *latencyFile1; // Declare latencyFile globally

double getCurrentTimeMillis() {
    return (double) clock() / CLOCKS_PER_SEC * 1000;
}

cv::Mat captureVideoFrame(cv::VideoCapture &cap) {
    cv::Mat frame;
    cap >> frame;
    return frame;
}

void *captureTask(void *arg) {
    // Set capture task to higher priority
    struct sched_param capture_param;
    capture_param.sched_priority = sched_get_priority_max(SCHED_FIFO); // Higher priority
    if (sched_setscheduler(0, SCHED_FIFO, &capture_param) == -1) {
        perror("Failed to set capture task priority");
        exit(EXIT_FAILURE);
    }
    cv::VideoCapture cap(0);
    if (!cap.isOpened()) {
        perror("Failed to open video capture");
        exit(EXIT_FAILURE);
    }

    while (1) {
```
// Measure latency — Start time
double start_time = getCurrentTimeMillis();
cv::Mat currentFrame = captureVideoFrame(cap);

pthread_mutex_lock(&frameMutex);

videoFrame = currentFrame.clone();

pthread_mutex_unlock(&frameMutex);

double end_time = getCurrentTimeMillis();
// Calculate and print latency
double latency = end_time - start_time;

// Write latency to file
fprintf(latencyFile1, "%.2f\n", latency);
fflush(latencyFile1); // Flush the file stream to ensure immediate write to disk

usleep(100000); // 10 milliseconds sleep for capture rate control

// Cleanup (optional)
cap.release();
pthread_exit(NULL);

void *sendTask(void *arg) {
    // Set send task to lower priority
    struct sched_param send_param;
    send_param.sched_priority = sched_get_priority_max(SCHED_FIFO) - 1; // Lower priority
    if (sched_setscheduler(0, SCHED_FIFO, &send_param) == -1) {
        perror("Failed to set send task priority");
        exit(EXIT_FAILURE);
    }

    while (1) {
        // Measure latency — Start time
double start_time = getCurrentTimeMillis();

        pthread_mutex_lock(&frameMutex);

cv::Mat currentFrame = videoFrame.clone();

        pthread_mutex_unlock(&frameMutex);

        // Measure latency — Start time
        // double start_time = getCurrentTimeMillis(); // Duplicate line, remove this

        std::vector<uchar> buffer;

cv::imencode(".jpg", currentFrame, buffer);

        ip_header.tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr) + buffer.size());

        memcpy(frame + sizeof(struct ether_header) + sizeof(struct iphdr) +
               sizeof(struct udphdr), buffer.data(), buffer.size());

        if (sendto(sockfd, frame,
                   sizeof(struct ether_header) + sizeof(struct iphdr) +
                   sizeof(struct udphdr) + buffer.size(), 0, (struct sockaddr *)&sa, sizeof(struct sockaddr_in)) == -1) {
            perror("Failed to send frame");
            exit(EXIT_FAILURE);
        }

        // Measure latency — End time
double end_time = getCurrentTimeMillis();
double latency = end_time - start_time;

// Write latency to file
fprintf(latencyFile, "%.2f\n", latency);
fflush(latencyFile); // Flush the file stream to ensure immediate write to disk

printf("Frame sent successfully\n");
usleep(10000); // 10 milliseconds sleep for send rate control

// Cleanup (optional)
// close(sockfd); // Remove this line, close the socket in the cleanup
// section of main() fclose(latencyFile); // Remove this line, close the file
// in the cleanup section of main()
pthread_exit(NULL);

int main() {
    // Create a socket
    sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
    if (sockfd == -1) {
        perror("Failed to create socket");
        exit(EXIT_FAILURE);
    }

    memset(&sa, 0, sizeof(struct sockaddr_ll));
    sa.sll_family = AF_PACKET;
    sa.sll_protocol = htons(ETH_P_ALL);
    sa.sll_ifindex = if_nametoindex("enp0s31f6"); // Replace with your interface name

    // Initialize Ethernet header
    struct ether_addr source_mac;
    struct ether_addr dest_mac;
    ether_aton_r("3C:52:82:43:B2:29", &source_mac);
    ether_aton_r("3C:52:82:43:A2:C8", &dest_mac);
    memcpy(eth_header.ether_shost, &source_mac, ETH_ALEN);
    memcpy(eth_header.ether_dhost, &dest_mac, ETH_ALEN);
    eth_header.ether_type = htons(ETH_P_IP); // IP packet

    // Initialize IP header
    ip_header.version = 4;
    ip_header.ihl = 5;
    ip_header.tos = 0;
    ip_header.id = 0;
    ip_header.ttl = 64;
    ip_header.proto = IPPROTO_UDP;
    ip_header.check = 0; // Set the checksum to 0 for simplicity
    ip_header.saddr = inet_addr("192.168.4.45"); // Replace with your source IP address
    ip_header.daddr = inet_addr("192.168.4.25"); // Replace with your destination IP address

    // Initialize UDP header
    udp_header.src_port = htons(200); // Replace with the desired source port
    udp_header.dst_port = htons(800); // Replace with the desired destination port
    udp_header.check = 0; // Set the checksum to 0 for simplicity

    // Construct the frame with IP, UDP, and video frame
    memcpy(frame, eth_header, sizeof(struct ether_header));
    memcpy(frame + sizeof(struct ether_header), &ip_header, sizeof(struct iphdr));
    memcpy(frame + sizeof(struct ether_header) + sizeof(struct iphdr),
           &udp_header, sizeof(struct udphdr));

    // Open the file for writing latency data
    latencyFile = fopen("./latency.txt", "w");
    if (latencyFile == NULL) {
        perror("Failed to open latency file");
        exit(EXIT_FAILURE);
    }

    // Write latency to file
    fprintf(latencyFile, "%.2f\n", latency);
    fflush(latencyFile); // Flush the file stream to ensure immediate write to disk

    // Close the file
    fclose(latencyFile);

    // Cleanup (optional)
    close(sockfd); // Remove this line, close the socket in the cleanup
    free(frame);
    free(eth_header);
    free(ip_header);
    free(udp_header);

    return 0;
}
Listing 1: NODE A Code

```c
 perror("Failed to open file");
 exit(EXIT_FAILURE);
}

latencyFile1 = fopen("./latency1.txt", "w");
if (latencyFile1 == NULL) {
  perror("Failed to open file");
  exit(EXIT_FAILURE);
}

// Initialize mutex
pthread_mutex_init(&frameMutex, NULL);

// Create threads for capture and send tasks
pthread_t captureThread, sendThread;
pthread_create(&captureThread, NULL, captureTask, NULL);
pthread_create(&sendThread, NULL, sendTask, NULL);

// Main thread can also perform some other tasks or wait for threads to finish
// Wait for threads to finish (optional)
pthread_join(captureThread, NULL);
pthread_join(sendThread, NULL);

// Cleanup
close(sockfd);
fclose(latencyFile);
fclose(latencyFile1);
pthread_mutex_destroy(&frameMutex);
return 0;
```
This code is specifically developed for a 4G modem and comprises two essential tasks: the “processTask”, which has a lower priority responsible for processing frames received from NODE A and converting/decoding them into the 4G format, and the “receiveTask”, which has the highest priority accepts frames from NODE A and computes the latency over the TSN. Overall, the code facilitates bidirectional data communication using both client and server functionalities.

```c
#include <net/if.h>
#include <netinet/ether.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <opencv2/opencv.hpp>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <unistd.h>

#define FRAME_SIZE 1500

struct ThreadData {
    int sockfd_receive;
    int sockfd_send;
    FILE *communication_latency_file;
    pthread_mutex_t frame_mutex;
    cv::Mat received_frame; // shared data protected by mutex
};

double getCurrentTimeMillis();
cv::Mat decodeVideoFrame(const uint8_t *data, size_t size);
void *receiveTask(void *data);
void *processTask(void *data);

int main() {
    int sockfd_receive;
    struct sockaddr_ll sa;
    pthread_mutex_t frame_mutex = PTHREAD_MUTEX_INITIALIZER;
    char frame[FRAME_SIZE];

    // Create a raw socket
    sockfd_receive = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
    if (sockfd Receive == -1) {
        perror("Failed to create socket");
        exit(EXIT_FAILURE);
    }

    // Set the network interface index
    memset(&sa, 0, sizeof(struct sockaddr_ll));
    sa.sll_family = AF_PACKET;
    sa.sll_protocol = htons(ETH_P_ALL);
    sa.sll_ifindex = if_nametoindex("enp0s31f6"); // Replace "enp0s31f6" with your interface name

    // Bind the receiving socket to the network interface
    if (bind(sockfd_receive, (struct sockaddr *)&sa,
             sizeof(struct sockaddr_ll)) == -1) {
        perror("Failed to bind socket");
        close(sockfd_receive);
        exit(EXIT_FAILURE);
    }

    // Open file for saving communication latency
    FILE *communication_latency_file =
            fopen("communication_latency_intermediate.txt", "w");
```
if (communication_latency_file == NULL) {
    perror("Failed to open latency file");
    close(sockfd_receive);
    exit(EXIT_FAILURE);
}

// Initialize thread data
struct ThreadData threadData;
threadData.sockfd_receive = sockfd_receive;
threadData.communication_latency_file = communication_latency_file;
threadData.frame_mutex = frame_mutex;

// Create threads
pthread_t receiveThreadId, processThreadId;
pthread_create(&receiveThreadId, NULL, receiveTask, &threadData);
pthread_create(&processThreadId, NULL, processTask, &threadData);

// Sleep for a while to allow threads to run (replace with your main logic)
sleep(10);

// Cancel threads
pthread_cancel(receiveThreadId);
pthread_cancel(processThreadId);

// Join threads
pthread_join(receiveThreadId, NULL);
pthread_join(processThreadId, NULL);

// Close the file
fclose(communication_latency_file);

// Close the socket
close(sockfd_receive);

return 0;
}

double getCurrentTimeMillis()
{
    return (double)clock() / CLOCKS_PER_SEC * 1000;
}

cv::Mat decodeVideoFrame(const uint8_t* data, size_t size)
{
    cv::Mat decoded_frame = cv::imdecode(
        cv::Mat(1, size, CV_8U, const_cast<uint8_t*>(data)), cv::IMREAD_COLOR);
    return decoded_frame;
}

void* receiveTask(void* data) {
    struct ThreadData* threadData = (struct ThreadData*)data;
    int sockfd_receive = threadData->sockfd_receive;
    FILE* communication_latency_file = threadData->communication_latency_file;

    while (1) {
        // Measure start time for communication latency
        double communication_start_time = getCurrentTimeMillis();

        char frame[FRAME_SIZE];
        ssize_t recv_len = recv(sockfd_receive, frame, FRAME_SIZE, 0);
        if (recv_len == -1) {
            perror("Failed to receive frame");
            close(sockfd_receive);
            fclose(communication_latency_file);
            exit(EXIT_FAILURE);
        }

        // Measure end time for communication latency
        double communication_end_time = getCurrentTimeMillis();

        // Calculate communication latency
        double communication_latency =

```
communication_end_time – communication_start_time;

// Protect the shared data (received frame) using a mutex
pthread_mutex_lock(&threadData->frame_mutex);
threadData->received_frame = decodeVideoFrame(frame, recv_len);
pthread_mutex_unlock(&threadData->frame_mutex);

fprintf(communication_latency_file, "%.2f\n", communication_latency);
fflush(communication_latency_file);
}
return NULL;

void ∗processTask(void ∗data) {

struct ThreadData ∗threadData = (struct ThreadData ∗) data;

// Open the socket for sending outside the loop
int send_sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (send_sockfd == -1) {
    perror("Failed to create send socket");
    exit(EXIT_FAILURE);
}

struct sockaddr_in dest_addr;
memset(&dest_addr, 0, sizeof(dest_addr));
dest_addr.sin_family = AF_INET;
dest_addr.sin_port = htons(100);

if (inet_aton("192.168.191.234", &dest_addr.sin_addr) == 0) {
    perror("Invalid destination IP address");
    close(send_sockfd);
    exit(EXIT_FAILURE);
}

while (1) {

    // processing latency
double processing_start_time = getCurrentTimeMillis();

    // Protect the shared data (received frame) using a mutex
    pthread_mutex_lock(&threadData->frame_mutex);
    // Assuming the received frame is in threadData->received_frame
    cv::Mat frame_to_send = threadData->received_frame.clone();
    pthread_mutex_unlock(&threadData->frame_mutex);

    // since the frame is an image
    std::vector<uint8_t> frame_data;
    cv::imencode(".jpg", frame_to_send, frame_data);

    ssize_t send_len =
        sendto(send_sockfd, frame_data.data(), frame_data.size(), 0,
               (struct sockaddr *)&dest_addr, sizeof(dest_addr));
    if (send_len == -1) {
        perror("Failed to send frame");
        close(send_sockfd);
        exit(EXIT_FAILURE);
    }

    // processing latency
double processing_end_time = getCurrentTimeMillis();

double processing_latency = processing_end_time – processing_start_time;

    fprintf(threadData->processing_latency_file, "%.2f\n", processing_latency);
    fflush(threadData->processing_latency_file);
}

// Close the socket after the loop
close(send_sockfd);

}
return NULL;

Listing 2: NODE B Code
This code is designed to handle frame reception from NODE B. It features a “receiveTask,” implemented as a server with the highest priority. This task receives frames from NODE B, calculates the latency over the 4G network, and ensures seamless data reception. Additionally, there is a “processTask” responsible for further processing the received frames and displaying them. The code effectively manages frame processing and latency calculation in the context of 4G communication.

```c
#include <arpa/inet.h>
#include <netinet/in.h>
#include <opencv2/opencv.hpp>
#include <pthread.h>
#include <sched.h>
#include <stdlib.h>
#include <string.h>
#include <sys/select.h>
#include <sys/socket.h>
#include <time.h>
#include <unistd.h>

#define BUFFER_SIZE 1500

pthread_mutex_t communication_latency_mutex = PTHREAD_MUTEX_INITIALIZER;

struct ThreadData {
    int sockfd;
    FILE *processing_latency_file;
    FILE *communication_latency_file;
    cv::Mat received_frame; // Mutex-protected received frame
    pthread_mutex_t frame_mutex;
};

void * receiveTask ( void * data ) {
    // ... (receiveTask's implementation)
}

void * processTask ( void * data ) {
    // ... (processTask's implementation)
}

int main() {
    int sockfd;
    struct sockaddr_in addr;
    int port = 100; // Use the same port as the sender code
    const char *ip_address = "192.168.191.234"; // Use the same IP address as the sender code

    // Create a UDP socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if ( sockfd == -1 ) {
        perror("Failed to create socket");
        exit(EXIT_FAILURE);
    }

    // Set up the address structure
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    if ( inet_pton(AF_INET, ip_address, &addr.sin_addr) <= 0 ) {
        perror("Invalid address");
        close(sockfd);
        exit(EXIT_FAILURE);
    }

    // Bind the socket to the address and port
    if ( bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) == -1 ) {
        perror("Failed to bind socket");
        close(sockfd);
        exit(EXIT_FAILURE);
    }

    // ... (main's implementation)
}
```

C Appendix 3
// Open files for saving latency
FILE *processing_latency_file = fopen("processing_latency.txt", "w");
FILE *communication_latency_file = fopen("communication_latency.txt", "w");
if (processing_latency_file == NULL || communication_latency_file == NULL)
{
    perror("Failed to open latency files");
    close(sockfd);
    exit(EXIT_FAILURE);
}

// Initialize thread data
struct ThreadData threadData;
threadData.sockfd = sockfd;
threadData.processing_latency_file = processing_latency_file;
threadData.communication_latency_file = communication_latency_file;
pthread_mutex_init(&threadData.frame_mutex, NULL); // Initialize the mutex

// Create threads
pthread_t receiveThreadId, processThreadId;
pthread_create(&receiveThreadId, NULL, receiveTask, &threadData);
pthread_create(&processThreadId, NULL, processTask, &threadData);

// Sleep for a while to allow threads to run (replace with your main logic)
sleep(10);

// Cancel threads
pthread_cancel(receiveThreadId);
pthread_cancel(processThreadId);

// Join threads
pthread_join(receiveThreadId, NULL);
pthread_join(processThreadId, NULL);

// Destroy the mutex
pthread_mutex_destroy(&threadData.frame_mutex);

// Close the files
fclose(processing_latency_file);
fclose(communication_latency_file);

// Close the socket
close(sockfd);

return 0;

}

double getCurrentTimeMillis() {
    return (double)clock() / CLOCKS_PER_SEC * 1000;
}

cv::Mat decodeVideoFrame(const uint8_t *data, size_t size) {
    cv::Mat decoded_frame = cv::imdecode(
        cv::Mat(1, size, CV_8U, const_cast<uint8_t*>(data)), cv::IMREAD_COLOR);
    return decoded_frame;
}

void *receiveTask(void *data) {
    struct ThreadData *threadData = (struct ThreadData *)data;
    int sockfd = threadData->sockfd;
    FILE *communication_latency_file = threadData->communication_latency_file;
    struct sched_param param;
    param.sched_priority = sched_get_priority_max(SCHED_FIFO);
    pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);

    while (1) {
        fds_set readSet;
        FD_ZERO(&readSet);
        FD_SET(sockfd, &readSet);
        struct timeval timeout;
double communication_start_time = getCurrentTimeMillis();
int ready = select(sockfd + 1, &readSet, NULL, NULL, &timeout);
if (ready > 0 && FD_ISSET(sockfd, &readSet)) {
    char buffer[BUFFER_SIZE];
    ssize_t recvLen = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, NULL, NULL);
    if (recvLen == -1) {
        perror("Failed to receive packet");
        close(sockfd);
        exit(EXIT_FAILURE);
    }
    // Decode the received video frame
    cv::Mat received_frame = decodeVideoFrame(reinterpret_cast<uint8_t*>(buffer), recvLen);
    double communication_end_time = getCurrentTimeMillis();
    double communication_latency = communication_end_time - communication_start_time; // in milliseconds
    // Lock the mutex before updating the shared frame
    pthread_mutex_lock(&threadData->frame_mutex);
    threadData->received_frame = received_frame.clone();
    pthread_mutex_unlock(&threadData->frame_mutex);
    printf("communication_latency_file", ".2f\n", communication_latency);
    fflush("communication_latency_file");
}
return NULL;
}

void *processTask(void *data) {
    struct ThreadData *threadData = (struct ThreadData *)data;
    FILE *processing_latency_file = threadData->processing_latency_file;
    struct sched_param param;
    param.sched_priority = sched_get_priority_max(SCHED_FIFO) - 1;
    pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);
    while (1) {
        double processing_start_time = getCurrentTimeMillis();
        // Lock the mutex before accessing the shared frame
        pthread_mutex_lock(&threadData->frame_mutex);
        cv::Mat received_frame = threadData->received_frame.clone();
        pthread_mutex_unlock(&threadData->frame_mutex);
        // Display the received frame (replace with actual display logic)
        cv::imshow("Received Frame", received_frame);
        cv::waitKey(1);
        // Save processing latency to file
        double processing_end_time = getCurrentTimeMillis();
        double processing_latency = processing_end_time - processing_start_time;
        printf("processing_latency_file", ".2f\n", processing_latency);
        fflush("processing_latency_file");
    }
    return NULL;
}

Listing 3: NODE C Code