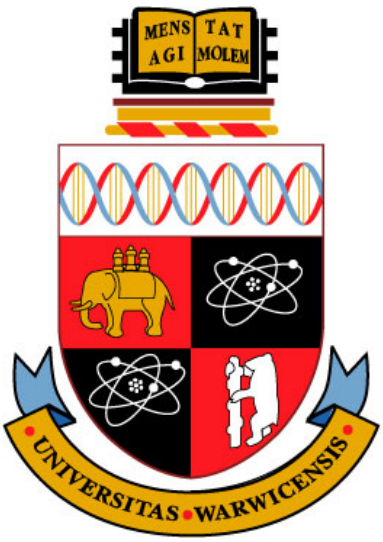
WARWICK
THE UNIVERSITY OF WARWICK



Towards Reliable Logging in the Internet of Things Networks

by

Sara Alhajaili

Thesis

Submitted to the University of Warwick

in partial fulfilment of the requirements

for admission to the degree of

Doctor of Philosophy in Computer Science

Department of Computer Science

September 2022

Contents

List of Tables

List of Figures

List of Algorithms

Acknowledgments

First and foremost, I offer my utmost gratitude to Allah for providing me with this opportunity and granting me the fortitude and determination to accomplish this journey.

I am also deeply grateful to Dr Arshad, my supervisor, for his exceptional guidance and mentorship throughout this endeavour. His support and encouragement have been invaluable through this long journey and the never-ending deadlines.

I like to thank all those who have assisted me throughout my academic journey, including the Department of Computer Science, my lab colleagues, and especially Jawaher. The support and patience I have received have been invaluable in helping me overcome research obstacles and writing challenges.

I'd also like to thank the University of Jeddah and, on its behalf, the Saudi Higher Education Ministry for funding my research and providing me with all the resources I needed to finish the prerequisites for this degree.

Finally, and most importantly, I want to express my utmost gratitude and appreciation to my family. They have always been my rock, supporting me and encouraging me to pursue my dreams. I am incredibly grateful to my parents, Salah and Muneera, who instilled in me the values of perseverance and determination. My siblings, Raghad and Ammar, have been my confidants and sounding boards, always there for me when I needed them. And Anas, my partner in crime, thank you for being the yin to my yang and challenging me to be the best version of myself. Last but not least, my niece Reema, you have brought so much joy and light into my life. Thank you for being my family, for your unwavering love and support, and for making this journey much more bearable.

Declarations

The author confirm that this thesis has not been submitted for a degree at another university. In addition, the author has previously published parts of this thesis or submitted them for review.
Parts of this thesis have been previously published by the author in the following:

- Alhajaili, Sara, and Arshad Jhumka. *"Auditability: An Approach to Ease Debugging of Reliable Distributed Systems."* 2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC). IEEE, 2019.
- Alhajaili, Sara, and Arshad Jhumka. *"Reliable Logging in Wireless IoT Networks in the Presence of Byzantine Faults."* 2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). IEEE, 2021.

In process:

- Alhajaili, Sara, and Arshad Jhumka. *"MAC Platform Effects on Reliable Logging in Internet of Things (IoT) Networks"* Journal of Parallel and Distributed Computing Special Edition: Dependability and Scalability of Distributed Systems in the Presence of Faults and Security Threats.

Abstract

The internet of things is one of the most rapidly developing technologies, and its low cost and usability make it applicable to various critical disciplines. Being a component of such critical infrastructure needs, these networks have to be dependable and offer the best outcome. Keeping track of network events is one method for enhancing network reliability, as network event logging supports essential processes such as debugging, checkpointing, auditing, root-cause analysis, and forensics.

However, logging in the IoT networks is not a simple task. IoT devices are positioned in remote places with unstable connectivity and inadequate security protocols, making them vulnerable to environmental flaws and security breaches.

This thesis investigates the problem of reliable logging in IoT networks. We concentrate on the problem in the presence of Byzantine behaviour and the integration of logging middleware into the network stack. To overcome these concerns, we propose a technique for distributed logging by distributing loggers around the network. We define the logger selection problem and the collection problem, and show that only the probabilistic weak variant can solve the problem. We examine the performance of the Collector algorithm in several MAC setups.

We then explore the auditability notion in IoT; we show how safety specification can be enforced through the analogies of fair exchange.

Next, we review our findings and their place in the existing body of knowledge. We also explore the limits we faced when investigating this problem, and we finish this thesis by providing opportunities for future work.

Acronyms

CPU Central Processing Unit.

CSMA Carrier Sense Medium Access.

DAG Directed Acyclic Graph.

DAO Destination Advertisement Object..

DIO Destination-Oriented Directed Acyclic Graph Information Object..

DIS Destination-Oriented Directed Acyclic Graph Information Solicitation.

DODAG Destination-Oriented Directed Acyclic Graph..

ETX Expected Transmission Count.

FDMA Frequency Division Multiple Access.

FIFO First In First Out.

IoTWSN Internet of Thing Wireless Sensor Based Networks..

LSP Logger Selection Problem.

MAC Medium Access Control..

MSF Minimum Scheduling Function.

RPL The IPv6 Routing Protocol for Low-Power and Lossy Networks..

TDMA Time Division Multiple Access.

TSCH Time Slotted Channel Hopping.

Symbols

Γ	A property.
ρ	A safety property.
$\alpha, \varrho, \chi, \kappa, \eta, \sigma$	A trace.
δ	A distance between nodes.
G	A graph.
V	Set of vertices.
E	Set of edges.
γ	Number of nodes in V.
β	Set of events.
e	An event.
ω	A predicate
v	A variable.
p	A process.
ch	A communication channel.
$\mathcal{P}$	A program.
s	A state.
$\mathcal{S}$	A set of states.
ϵ	State space.
$\mathcal{I}$	Initial State.
$\mathcal{C}$	A set of clusters.
$\mathcal{L}$	A set of loggers.
l_i	Logger i.
l	Log entry.
$\mathcal{CM}$	A control message.
$\mathcal{DM}$	A data message.

seq	Message Sequence Number.
$\mathcal{UL}$	Unconfirmed Log.
$\mathcal{CL}$	Confirmed Log.
$buff$	A node Buffer.
$buff\_size$	A node buffer size.
k, f	Faulty nodes.

Chapter 1

Introduction

The Internet of Things wireless sensors-based networks (IoTWS) are becoming increasingly popular due to their simplicity of implementation, affordability, and scalability [68] [159]. The traits that highlight the devices that comprise these networks - the nodes - are the causes of these benefits. IoTWS [1]. Nodes are compact, battery-powered devices with limited communication and computing capabilities, including wireless connection, memory, sensors, and actuators[167] [86] [178]. While these qualities enable the flexibility of IoTWS network installations, they also expose the network to cyber-attacks and environmental impacts [14] [169] [168], see Figure 1.1.

The nodes are programmed to gather data about their surroundings, transmit notifications, and receive commands, allowing them to engage with a remote system or human. As a result, they are prevalent in the modern world, ranging from the industrial sector, where they are used to regulate and monitor production processes[32], to public environmental monitoring [53], where they are used to monitor the health of infrastructure, to personal health monitoring [100] [143] and smart home applications [92] [193].

Due to their unique characteristics, they are used as a basis for various safety-critical systems where nodes in these networks are programmed to react based on their readings. Many IoT applications utilise wireless sensor networks as their base to serve in industrial settings[183], healthcare [209], military [19] and agricultural [206] environments. Nodes in these applications can react according to the data they sense from the environment, which is exchanged throughout the network. For example, nodes can sense the temperature, and based on the sensor readings, the actuators can either decrease or increase the temperature, allowing autonomous control of the environment.

[1] In the remainder of this work, we will use the terms wireless sensor network, internet of things and internet of things wireless sensor-based network interchangeably

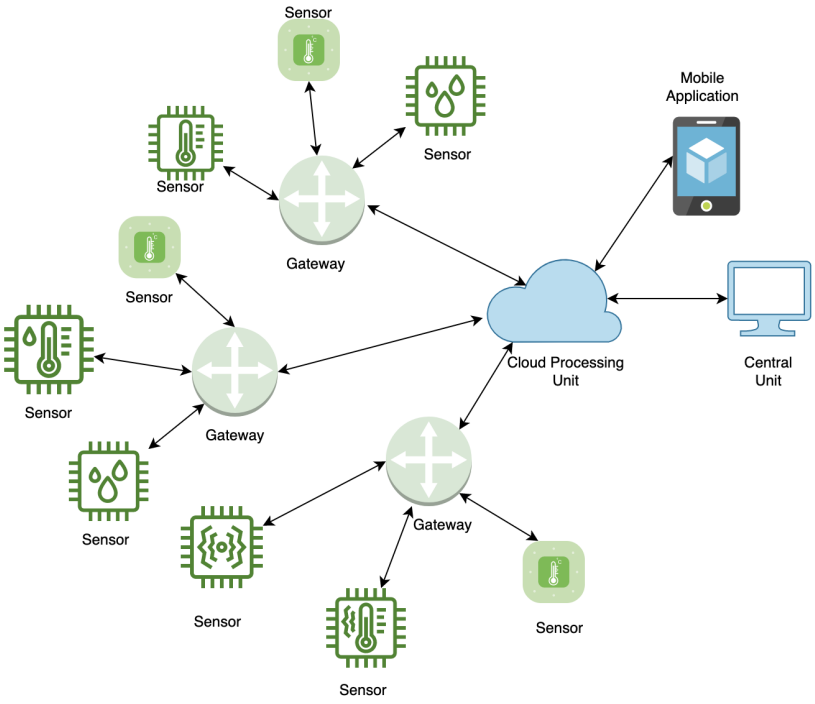


Figure 1.1: Internet of Things Wireless Sensor Based Network Structure

One significant example is that of healthcare applications [71] [209], where wearable body sensors are used to monitor and control patients' health. Nodes in these applications can sense heart beat rate, blood pressure and body temperature along with other metrics and based on these readings, they are programmed to perform some action [64]. This type of network is known as Body Area Network (BAN) or Body Sensor Network (BSN) [149].

Another example of these safety-critical systems is their industrial applications. In these applications, the physical entities responsible for production are monitored and controlled in real-time by deploying WSN and computer-based algorithms [139].

Because of these applications' critical nature, they require reliable IoTWS networks. However, it is difficult to achieve this objective due to the limitations of the nodes that undermine these networks.

Moreover, to reduce costs, IoTWS networks are generally assessed in a simulated environment [34] [33], allowing researchers to ensure their applications' functionality and troubleshoot problems before deployment. Simulation settings, however, provide ideal testing conditions; networks are always reliable, and all data is transmitted and received appropriately. That is not the case with actual networks; deployments are typically subject to severe environmental interference and unreliable links, are prone to failures and crashes and are especially susceptible to attacks.

By design, IoT wireless sensor-based networks are set up as open networks; simultaneously, they are valuable tools for monitoring other surroundings. They are challenging to monitor, and as a natural consequence, establishing the accuracy of historical events takes time and effort. In order to correctly keep history logs, logs sent throughout the network need to be kept intact so that they can show any inconsistency in the network.

Since they are used widely in safety-critical systems, IoTWS networks implement various techniques to provide reliable communications to avoid faults due to communication issues. Implementing the medium access control [51] [21] network layer, which is the layer that controls the message flow in an IoTWS network, is one way of mitigating these issues. As these networks use a broadcasting communication range, the MAC layer decides which messages to receive and which to drop; it is also the layer that is responsible for sending messages.

Research then goes further to enhance the reliability of the MAC layer by introducing the time-division multiple access, TDMA [179] protocol, which is a medium access protocol that divides the time of the network into several time slots, allowing each node their slot to transmit, thereby minimising conflict in the network. For added control and to avoid collisions and message loss, another variation of TDMA is introduced, which is the time-division channel hopping [202] MAC protocol. Besides allocating each node a time slot for transmission, it also allocates a specific radio channel to transmit on, increasing the network reliability and throughput while decreasing the energy consumption of the nodes.

However, one consequence of these protocols is that they present different scenarios for malicious nodes to masquerade, adding to the fault models of the system. Thus, while these protocols increase network reliability and dependability, they introduce several challenges regarding the reliability and security of the network. Nodes in these protocols turn their radio on only when it is time for them to send or receive a message; when nodes fail to send or do not receive a message, it is challenging to know whether this happened because the node's radio was off or whether the node was acting maliciously.

1.1 Motivation

At some point during their operation, all computing systems create logs. Logs are critical components in the life cycle of any system, providing a record of system execution history as the foundation for many system maintenance processes.

For example, systems use logs to construct checkpoints that allow the system to recover from a fault or crash [203] [110] [111]. They are helpful in debugging

since they help developers to understand what went wrong and how to solve it[184] [58]. They also serve as the foundation for forensic investigations since they provide evidence and records of earlier instances [30] [141].

Nevertheless, while logging has been an element of computational systems since its early beginnings, and while several studies have been conducted on how to execute logging in traditional systems, more research should be conducted on logging in IoTWS networks.

IoTWS network presents several challenges for the event logging problem that does not exist in other areas of distributed systems. IoTWS networks are open and distributed networks with unpredictable connections and unreliable links; they are also exposed to threats and environmental effects.

These properties make central logging problematic; logs must travel vast distances across insecure and unreliable connections, increasing the likelihood of message loss. Nodes cannot log locally owing to memory limits, nor can they push logs to secure storage regularly due to bandwidth and energy constraints.

Logging is also done as a by-product of applications or algorithms in the IoT networks; there are few rules governing these logging processes, and the problem needs to be adequately defined within the IoTWS network area. Moreover, while logs can be easily generated and verified in a correctly functioning network with no malicious behaviour, the case is different in the presence of Byzantine behaviour. Additionally, IoTWS networks are open networks that can be readily compromised, making logging network events complicated.

As IoTWS applications dominate the safety-critical system base, recording network executions are becoming increasingly important for various reasons. Logs can help to determine what happened, when, how, and who was responsible. The logs may be used to diagnose execution errors, generate checkpoints for recovering from faults and network crashes, and monitor network performance to understand better the network's behaviour and deal with problems that may arise over time.

Moreover, logs are critical components of secure systems for detecting malicious activity or abnormal system behaviour, which is essential to be efficiently conducted, considering the nature of the safety-critical systems. Finally, logs are a core part of the system auditing process; auditors may establish how network resources were used using these logs.

Generally, logging in IoTWS networks occurs during or after the execution of a transaction. Because the memory capacity of IoTWS devices is insufficient to keep significant logs, logs are often routed to a central unit for collection.

Cryptography is one of the approaches used to ensure the messages' security and integrity. Logs are created locally on the nodes, and then, depending on the

application, they are sent to a central collecting point. Logs can be modified, lost or omitted during the transaction through malicious nodes or network faults. However, the cryptography algorithms used in IoTWSN are mostly hardware-based or a lighter software version that requires more computation [164].

1.2 Why Correct Logs?

Internet of Things wireless sensor-based networks are implemented in several contexts with varying levels of criticality; thus, it is vital to validate network events. The logs created by these networks are utilised for debugging, auditing, checkpointing, and forensic investigations to study the network's history. Why is a system that creates a second set of logs or histories needed if the network application can generate them?

This is a highly relevant question; however, while network applications create varied histories by design or as a by-product of their execution, there is no formal technique for validating their authenticity.

Any node can create logs, which can be edited and erased; in IoTWS networks, this can occur without leaving a trace unless a cryptographic primitive is utilised. Thus, If cryptography can address this problem, why is it not utilised?

Although cryptography is undeniably valuable for safeguarding the messages' security and integrity [174] [128] [152], we acknowledge that it may not offer a complete solution to all network-related problems. Problems such as faults, unreliable connections and logging the network events cryptography cannot solve it. Additionally, the process of implementing cryptography-based techniques can be both costly and time-intensive. In order to guarantee that our network logs retain their reliability and dependability, we have opted to investigate alternative methods that can work in conjunction with cryptography.

Blockchains [17] [194] are a technique for creating a chain of reliable logs while maintaining the records' original order, confidentiality and immutability. Given the limited resources available in an IoTWSN, the difficulties inherent in meeting the technology's demanding computing, storage, and communication needs become even more pronounced [18].

The notion of log correctness varies based on the application. For instance, while for one application, logging the events of the system makes the log correct, in another application, if these events are not in order, the log cannot be considered correct. Thus we can ensure the log's correctness by enforcing an application-specific safety specification.

Consequently, we gravitate towards an approach that permits us to generate

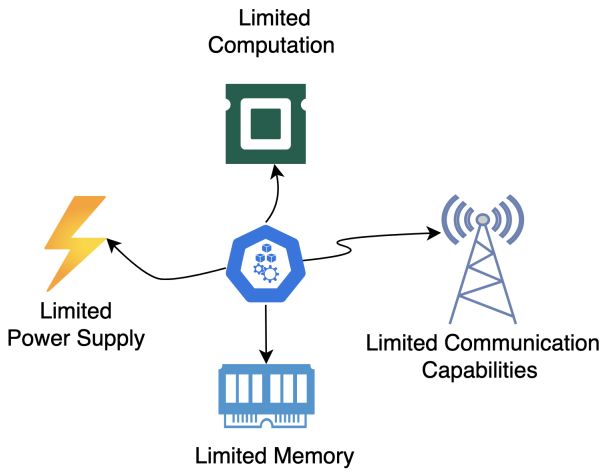


Figure 1.2: Internet of Things Wireless Sensor Based Network Constrains

logs in a way that is applicable and affordable in IoTWSN networks, that can withstand malicious actions in the network, maintains the log's integrity and leaves no doubt as to whether a certain event occurred.

1.3 Issues and Challenges

As mentioned earlier, nodes in an IoTWS network have minimum computation power and memory, which reduces their energy consumption and enables them to operate on batteries rather than power banks, contributing to their cost-effectiveness. Figure 1.2. Nevertheless, whilst these characteristics contribute to the cost-effectiveness of the IoTWS network, they also create a number of concerns and challenges with the logging problem.

The first question that comes to mind is where the logs will be gathered. Typically, logs are centrally collected in an IoT network and relayed to a remote server by nodes that periodically send logs to a base station or sink. If possible, logs are kept in permanent storage for an extended period of time. However, due to the high volume of logs generated in these networks, they are usually deleted after a short period.

Moreover, WSNs are often built to cover broad areas with many nodes dispersed at varying distances; this implies that packets, also termed logs, must travel a great distance to reach the sink, resulting in message loss and latency. Additionally, this method of log collection results in high energy use. Alternatively, collecting logs in a distributed manner might solve these issues. Unfortunately, this is not the case.

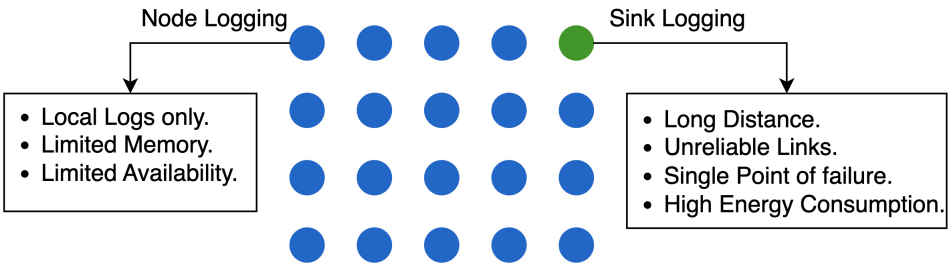


Figure 1.3: The Challenges of Node Level Logging and Sink Level Logging

IoTWS nodes are resource-constrained; consequently, for logs to be collected in a distributed manner, nodes must have the additional capacity to store and process these logs; otherwise, the cost-effectiveness of IoTWS would be compromised. Due to the environments in which IoTWS networks are often deployed, collecting logs in a distributed way may make it difficult to verify the availability of these logs, and log integrity is easily compromised. In other words, if the logs are stored in a distributed manner over the network, how can we ensure the integrity of each log? How can we ensure that obtaining the logs did not alter them? In addition, if the logs were unavailable, how could we determine whether the availability was compromised due to network issues or malicious behaviour?

Another challenge is that as the network expands, the distance between the nodes increases, and the topology will frequently change to cope with the new additions, compromising the network's reliability and making the network more susceptible to attacks; this indicates that the logs may be lost during the transaction or compromised by an adversary.

Furthermore, logs produced in the IoTWS applications and their algorithms assume that the node is correct, acting according to the system, or that the whole network is free from Byzantine behaviours; this is not easy to uphold in open networks such as the IoTWS networks. IoTWS networks are generally more vulnerable and prone to cyber-attacks, posing new challenges regarding recording system activities.

One major issue to consider in this context is that of power consumption. Can we collect logs reliably and securely so that the impact on the power consumption of the nodes can be mitigated?

It will be challenging to accomplish this without making certain network configuration adjustments. The medium access control (MAC) mechanism, which regulates node behaviour to minimise energy consumption, is one such configuration that plays a significant role in minimising node power consumption. Nevertheless, how do MAC protocols perform in the context of Byzantine faults in an open system

such as the IoT?

Various MAC protocols are utilised for the IoTWS networks, depending on the network's intended use. Some protocols schedule the node's up-time to reduce energy consumption and conflict, resulting in a more reliable network. In contrast, other protocols always keep the node's radio on and operational. However, this presents several issues when attempting to capture network events while considering malicious behaviour. Is the node offline due to a predetermined schedule? Or is it faulty due to Byzantine behaviour?

Furthermore, research in IoTWSN logging has focused mainly on the data collection element, with data collected centrally or pushed to cloud storage at a later stage [57]. There is no formalisation of the logging process, and more research needs to be focused on the state of the logs within the context of a malicious presence in the network.

1.4 Problem Statement

To ensure that wireless sensor-based networks in the Internet of Things function properly, creating a system that enforces proper node behaviour is essential. One way to achieve this is by developing a system that records events at every possible step; this incentivises nodes to behave correctly and allows malicious behaviour detection. The system can ensure the network runs smoothly by having logs available for debugging, fault detection, and eliminating malicious nodes. Previous research on distributed fault tolerance, particularly Byzantine fault tolerance, has emphasised the significance of such systems in guaranteeing proper node behaviour.

Thus, this thesis statement is: Reliable logging is achievable in the Internet of Things wireless sensor-based networks. In order to achieve this goal, we make the following contributions:

- We define and formalise the Logging Collection Problem by splitting it into two different problems: The Logger Selection and the Collection Problem.
- We present impossibility results in solving the logging collection problem within the presence of Byzantine faults.
- We introduce the Collector Algorithm with two versions, The Receiver Oriented and The Sender Oriented.
- We then present a study about the impact of different MAC protocols, particularly CSMA and TSCH, on the Logging Collection problem.
- We define Auditability within IoT and how to enforce it.

1.5 Structure

The remainder of this work is organised as follows:

- Chapter 2, the literature review covers a comprehensive analysis of previous research conducted on data collection and logging in wireless sensor-based networks of the Internet of Things. This review also highlights the major findings in distributed fault tolerance, specifically focusing on the research that delves into Byzantine fault tolerance.
- Chapter 3 introduces the systematic models used in the theoretical results and the technical model setups in terms of the operating systems, routing protocols and deployment configurations.
- Chapter 4 presents the formalisation of the problem statement
- Chapter 5 Introduce the collection problem, present the problem definitions and the *Collector* algorithm and study its performance in several MAC configurations.
- In chapter 6, we investigate the impact of different platforms on the logging problem.
- Chapter 7 introduces the auditability case study in the IoT networks.
- Chapter 8 explores the significance of this research's findings for the existing body of knowledge and the limitations observed during this investigation.
- Chapter 9 concludes the thesis and presents pointers for future work.

Chapter 2

Literature Review

This work is dependent on prior work in the internet of things wireless sensor based networks, byzantine fault tolerance, and fair exchange. Consequently, in this chapter we give a literature review of the relevant prior work in these areas.

2.1 Wireless IoT Networks

Wireless Internet of Things sensor networks consist of many small devices called nodes scattered through a wide area. These nodes have limited computation capacity, memory, and power; a frequently used standard node is the TelosB mote, which has a TIMSP430 microcontroller that only has 48KB of ROM and 10KB of RAM [4]. They also have transmitters that allow radio communications and other communication technologies such as Bluetooth and Zigbee.

To that end, most research in WSN focuses on decreasing the nodes' energy consumption to a minimum to keep the nodes alive as long as possible while keeping the network reliable and increasing its throughput, whether by optimising the network topology [37], routing [191] [201] [201] [173] or by radio duty cycling [60] [161] [52] while collecting data from the network.

However, these nodes have sensing and actuating abilities, which qualify them to be deployed in an intelligent environment where they are tasked with monitoring their surroundings and reporting their findings to a designated node. This designated node acts as a sink or central processing unit collects data from the network, analyses these data and makes intelligent decisions accordingly [175].

There are usually three types of nodes in the WSN: sink nodes, sensing nodes and router nodes. The sensor nodes have a sensing ability to collect information about their surroundings and send it to the sink or the base station. The sink or the base station then sends the collected message to a remote server or storage responsible for processing this information. It forwards the result to the end machine, whether

Figure 2.1: Wireless Sensor Networks Examples.

it is a lab top or mobile. On the other hand, the routing nodes are responsible for routing the messages from the sensing nodes to the sink [159] [182].

WSN is widely used for monitoring different domains: environmental, forest, health and green housing, see Figure 2.1. It is also used in the military, smart irrigation systems, pest management, industrial logistics and manufacturing process control. Additionally, it is used as the base of any IoT network or system, such as smart homes and smart living in general.

2.1.1 Nodes Attributes

The WSN has several capabilities in a limited scope that allows them to perform accordingly. These capabilities can be divided into:

- Sensing.
- Communications.
- Energy.

Sensing

The number of sensors in a WSN node depends on the application type. The most common are the temperature [59] [10], humidity [166], soil [155] and motion [134] sensors. Sensors can be inbuilt or attached externally; they convert the physical entities into electrical signals and send them to the CPU. The nodes' accuracy depends on the sensor's cost, precision and quality [124].

Communication

For the nodes to deliver their sensing data to the sink, they need a communication stack to connect to the network and participate in the data collection.

Communication is usually carried out through radio signals [20] [112] that correspond to the physical layer of the transmission process. The network stack in a wireless sensor node contains a physical data link, routing, transport and application layers.

Additionally, the data link layer includes the Medium Access Control (MAC)[210] [51]. The network layer is responsible for routing the data. Finally, the application layer is used to analyse the data sensed by the node.

Energy

Sensor nodes are usually self-powered; this means they are powered using either AA batteries or solar batteries [185]. Therefore, their energy reserves are limited, and considering their deployment locations, it is hard to maintain their energy supply and provide the required maintenance to refill it. Thus, one of the essential requirements when designing applications or protocols for WSNs is to use as little energy as possible.

2.1.2 Routing Protocols and Challenges

Routing protocols are a technique for exchanging routing information between routers and network nodes to construct a routing table, hence establishing routes between network nodes [148]. Using the proper routing protocol for the required system is crucial as it significantly affects the network's performance. Before considering the main routing algorithms in the WSN, we give a brief introduction to the parameters that we can apply to judge the effectiveness of the routing protocols.

2.1.2.1 Routing Challenges

Routing protocols are measured by their throughput, delivery ratio and latency [177]. Throughput refers to how many packets are sent from the sources and are received at their destination in a specific period [104]. At the same time, the delivery ratio is the number of packets sent that have been received. Finally, the latency is the communication delay, i.e. how long it takes for the packet to be delivered to its destination.

A suitable routing protocol will optimise the network throughput and keep the delivery ratio as high as possible and the latency to a minimum in all network scenarios.

Many routing protocols are widely used in WSN, based on well-known techniques proven to perform well in WSN, such as flooding, gossiping and collection trees.

Routing protocol faces several challenges [38] in WSN; here we present a brief list of these challenges that have influenced the result of this work:

- Nodes Deployments:
  As WSNs are usually deployed in broad areas, the environment in which they are deployed can hugely affect the network performance [115] [131] [136]. The position, climate and exposure to several environmental circumstances such as heat and humidity, affect node longevity and performance [159].

- Energy Consumption:
  As discussed previously, wireless sensor nodes have little energy reserves. Consequently, developing a reliable and secure routing protocol that does not deplete the node's energy is a significant task.

  One way to achieve these goals is clustering [31]. By dividing the network into several clusters and aggregating data through the clustering heads, the distance the packet travels and the up-time for the nodes are minimised. LEACH [96], FLOC [6], and HEAD [208] are some of the most popular clustering algorithms that are used and have been adapted for different WSN types [162].

  Another example is in [145], where authors investigate the possibility of predicting the data messages sent in WSN as another way to decrease energy consumption by reducing traffic. Using the ADAGA-P algorithm, they reduced the message traffic and increased the network's lifetime.

- Fault Tolerance:
  Wireless sensor networks are susceptible to numerous failure scenarios, including physical node failures such as crashes, battery depletion and cyber-attack failures. During the design phase of a routing protocol, various levels must contain mechanisms that allow the network to identify faults, recover from them and remain operational regardless of the faults [150] [180] [12].

- Security:
  IoTWSNs are responsible for sending sensitive data when they are used in remote locations. Keeping this information safe and making sure the data is available and the network connection is safe while keeping up with IoTWSN node capabilities is a significant challenge [207] [49] [39].

- Scalability:
  One of the fundamental IoTWSN design principles is ensuring these networks can scale without disrupting network processes. As the topology tends to fluctuate frequently, the node's links will be compromised, and much energy will be consumed. At the same time, these adjustments will necessitate an additional message exchange to accommodate them.

Designing a protocol that can cooperate with the IoTWS's scalability rate is challenging [81] [129] [163].

- Quality of Service
  A robust routing protocol must provide application-specific services such as network longevity, fault tolerance, converges and data accuracy, as well as end-to-end services such as low latency and packet loss and suitable bandwidth [94] [165].

- Data Reporting Module:
  Depending on the type of network [23], data reporting models in wireless sensor networks can be event-driven, time-driven, query-driven or hybrid. A routing protocol must be flexible enough to handle all these modules.

- Data Aggregation:
  IoTWSN collects data through aggregation at the sink eventually. Usually, this is done only at the sink or data is aggregated throughout the network by different nodes. A good aggregation model needs to deliver that data without any loss [116] [133] [8].

- Topology Maintains:
  Topology in WSN controls the network performance regarding latency, packet delivery ratio, link quality and energy consumption. Maintaining the topology to keep the network connection is vital in a well-designed routing protocol.

  Solving the problem of neighbourhood discovery in IoTWS networks is one way to save energy consumption and stabilise the network. In [160], the authors, for instance, explore the problem of neighbourhood discovery in IoTWSN. At the same time, the work presents promising results; however, they make no mention of whether it can perform as well in the presence of malicious behaviour.

As previously established, designing routing protocols for IoTWS networks is difficult; the criteria that must be considered are numerous, as is the necessity to continually deliver steady, reliable network performance.

2.1.2.2 Routing Protocols

WSNs are required to provide reliable, secure communication despite their energy [42], memory, and computation constraints [102]. Thus, designing routing protocols that can deliver to this standard without draining the node's resources is one of the significant essential components of IoT wireless sensor-based networks.

Routing protocols in WSN can be roughly categorised into hierarchical, flat and location-based routing [54], with the hierarchical being the most frequently used WSN.

RPL [192]. An IPv6 routing protocol for low-power lossy networks is one of the most frequently used protocols for routing in WSN. RPL build a directed acyclic graph DAG with the sink as the root, and then other nodes join the DAG by building their DAG, which is routed toward the first DAG, creating a distant oriented DAG, DODAG [117].

Other examples include clustering-based protocols such as the low energy adaptive clustering hierarchy LEACH [95], which is a single hop clustering with the cluster head a single hop away from the sink, draining much energy. On the other hand, the Energy Efficient Multihop Hierarchy routing EMHR [101] provides multihop clustering, decreasing the energy consumption of the clustering algorithm.

IoTWS network protocols are primarily based on a tree collection or a cluster-based approach to finally route data to the sink or the network's central unit.

2.1.3 Medium Access Control

The data link layer in the network stack dictates the medium access control (MAC). MAC is a crucial component in an IoTWS device network stack. It regulates channel access to various medium-sharing devices, ensures the network's reliability, flow control, and error detection and provides data framing [121].

There are two kinds of MAC protocols: (i) contention-based and (ii) contention-free [16]. Contention-based protocols require nodes to listen until the channel is free before sending and waiting for an acknowledgement. Since this takes the least amount of processing, these protocols are flexible and can be used in large networks.

However, contention-free MAC protocols give each node its time slot in a carefully designed manner to send or receive messages. The node then stays asleep until an agreed-upon time slot comes around.

We focus on contention-free MAC protocols in this study, notably the *Time Slotted Channel Hopping protocol (TSCH)* and associated schedulers: (i) *6TiSCH* [202] and (ii) *Orchestra* [77]; we now look at related work in that area of interest.

2.1.4 Time Slotted Channel Hopping MAC

This work uses the schedule-based MAC, mainly the *Time Slotted Channel Hopping - TSCH* MAC. TSCH is a variant of the Time Division Multiple Access MAC protocols *TDMA*, inspired by WirelessHART and ISA100.11a i. It divides the networks into time slots, with each time slot paired with a channel offset allowing the node to transmit on different frequencies.

A scheduler decides whether to transmit, receive, or sleep during each slot. Slots can be dedicated or shared slots, contention-free or with CSMA back-off (see Figure 3.5). TSCH schedulers are divided into two types: central schedulers and distributed schedulers [196]. Centralised schedulers are controlled by an operator and reserve network resources holistically. Decentralised schedulers, on the other hand, provide for greater adaptability and flexibility, which has sparked much interest in TSCH research [97].

Distributed TSCH schedulers are further classified into two types: collaborative and autonomous schedulers [197]. We used the autonomous scheduler Orchestra in this study; however, before we discuss Orchestra, we look at 6TiSCH, the first IPv6 over TSCH network implementation that serves as the foundation for numerous TSCH schedulers.

2.1.4.1 6TiSCH

6TiSCH implements IPv6 over the TSCH, allowing low-power lossy networks to utilise TSCH in their network stacks. 6TiSCH implementation consists of numerous steps, ranging from network formalisation [113] to slot allocation, with substantial research focused on optimising 6TiSCH performance. In this case, we examine 6TiSCH behaviour in the context of malicious behaviour.

In [44], the authors investigate the impact of malicious behaviour on network formalisation and, as a result, energy consumption in 6TiSCH networks.

While their results showed that malicious nodes in the network would not significantly affect small networks, their presence could affect large networks. Their effect is proportional to the number of malicious nodes and the size; the higher these parameters, the more disturbance these malicious nodes can make.

At the same time, the authors of [114] investigate the impact of DIS (DODAG Informational Solicitation) attack on network formation and find comparable results; malicious nodes can considerably increase network formation time and energy consumption without expending much energy.

In [45], the concept of selfishness is introduced. In a model related to the BAR model in the Byzantine fault tolerance field [15], the authors examine the presence of selfish nodes in the network. Nodes that do not collaborate do so because they wish to conserve their resources, not because they desire to engage in malicious behaviour.

Implementing selfish nodes in a 6TiSCH network leads to a significant increase in latency and a decrease in packet delivery ratio. However, they provide a new SA-6TiSCH algorithm that mitigates these network effects.

As far as we know, little research has been conducted into the effects of

Byzantine faults on reliable logging into the Internet of Things wireless sensor-based networks and how to mitigate those effects. However, much research has been conducted on the effects of Byzantine faults in these environments.

2.1.4.2 Orchestra

Orchestra is an autonomous TSCH scheduler for RPL networks, requiring no central scheduling, bargaining, or signalling. Each node's schedule is determined by its RPL state. Orchestra has a complete implementation in the Contiki-ng operating system [2] and is a widely known TSCH scheduler. Our system model presents a comprehensive explanation of Orchestra; hence, we focus on Orchestra performance within a Byzantine network.

In [137], the authors investigate TSCH and Orchestra performance under jamming attacks. Their findings indicate a visible decrease in the packet delivery ratio for both TSCH and Orchestra and a distinct rise in energy consumption.

2.2 Histories, Auditability and Logs

In computer science, execution history is a record of the state transitions of the system. A history is a collection of events, and individual event history can be referred to as a record or a log; they can also be saved as traces or replicas. Histories can also be recorded as global states of the system's execution [55].

Histories are generated in practically every system for various reasons, including debugging, monitoring, auditing, root cause analysis and, as a result, the problem of establishing and preserving these data. The latter has been an essential issue in the academic domain for several decades. However, while it has improved substantially in distributed systems, documenting the history of a network execution in the Internet of Things wireless sensor-based networks is still a work in progress, especially when considering what types of data we require and for what reason.

In this research, we look at reliably collecting logs for debugging, auditing and forensics applications. Simultaneously, there is a significant amount of work on data collection; when considering the how and assumptions, there is always the case of assuming that everything on the network is correct, that the nodes act according to the protocol, that the network does not encounter any malicious behaviour, or that no collection occurred while securing the network. Some works use a cryptographic approach; however, due to the nature of IoTWS network nodes, we decided against it.

The issues that arise when designing a tool to record these histories are related to (i) the ordering of the events between different processes [126], (ii) the overhead

that results from recording these histories [11] and (iii) the recording of a complete, consistent history [107]. We look at the data collection problem from a distributed computation perspective; then, we move to how it performs in WSN.

Falcon [154], for instance, is a logging tool for distributed systems that ensures the preservation of the happen-before relation between process and present visual representations of the logs. PeerReview [91] uses accountability to detect faults by requiring nodes to record their history of execution; then, utilising cryptography, nodes can replay these executions and assist in deciding whether the nodes are faulty. Full review [72], on the other hand, requires each node in the system to record logs to be audited by monitors in the network. The protocol uses incentives to discourage nodes from behaving maliciously.

While these approaches add overhead to the system and make assumptions about the communication model, these assumptions are reasonable for distributed systems models; however, that does not apply in the case of IoTWS networks.

Data collection [204] is a debugging technique widely used to achieve dependability in systems. Furthermore, it plays a significant role in creating checkpoints and snapshots to facilitate failure recovery. While the techniques to perform this task vary, they still need to be improved for IoT wireless sensor-based network settings because these systems must be equipped to execute this task flawlessly. These techniques are adapted from systems with reliable communication, central monitoring units and sophisticated security operations that handle malicious behaviour.

Data collection in IoT wireless sensor-based networks is conducted by relaying data to a sink designated to collect these data. While not tolerating Byzantine behaviour, the work of [84] does guarantee message delivery and reliability. This work presented the VIRTUS communication layer to overcome the unreliable nature of WSN by taking a modified version of the virtual synchrony concept initially presented by [40] for distributed systems. VIRTUS guarantees the delivery of the broadcast message to all or no recipients; in other words, an atomic broadcast is guaranteed. VIRTUS also ensures that the messages are delivered in the order they were sent.

In contrast, there are works such as TARDIS, a record and reply tool for debugging WSN. It records the state of the non-deterministic registries so that the network's execution can be replied to offline. As the authors demonstrated their tool's feasibility, they discovered a bug in the Collect Tree Protocol that is widely used in WSN.

Though the version presented by the authors in [189] is a reply of a single node, the tool could be extended to include multiple nodes; yet, similar to the case of VIRTUS, TARDIS does not tolerate Byzantine behaviour.

Clustering is also a data collection technique where cluster heads are distributed throughout the network to collect data from neighbouring nodes. Thus, minimising the distance a message travels and the node's time to stay up saves much energy [22].

It is also worth mentioning that clustering is a well-known problem to be NP-Complete [43] [176].

An NP problem is a problem that is accepted by a non-deterministic Turing machine in polynomial time [195]. A problem A' is reducible to problem A if, for every instance of A', there is an instance of A that can be constructed in polynomial bounded time such that solving an instance of A can solve an instance of A' as well. Thus, problem A is an NP-complete if we can prove that it can be solved by a non-deterministic Turing machine bounded by a polynomial length of the input, and there is a problem A' that is NP-complete that can be reduced to A [89]. Few clustering problems are polynomial-bounded; their number of computational steps is constrained by a polynomial in the problem's input length [48]. Thus, most clustering problems are considered NP-complete problems [122] [69].

While most IoTWS networks deployed in a fixed topology with the sink known to all the nodes at the start of the system, occasionally, mobile nodes are used in WSN, which causes the topology to change frequently, thus introducing a new challenge. Maintaining the routing consistency, the links' quality, and the nodes' power consumption levels in a changing topology is particularly challenging [188].

2.2.1 Auditability and Audit Logs

The Oxford Dictionary defines the audit process as "an official inspection of an organisation's accounts, typically by an independent body". It is a process performed either by internal examiners within the organisation or by external examiners appointed by the organisation in what is known as trusted third parties. In particular, information technology auditing involves "examining processes, IT assets, and controls at multiple levels within an organisation to determine the extent to which the organisation adheres to applicable standards or requirements"[87]. As businesses rely on the use of computing infrastructure to facilitate their operations and processes, auditing these systems is a crucial procedure of the operation of any organisation, not only to detect but also to prevent and correct incorrect behaviour [87].

In technology, audit logs have a wide range of usages in different fields. From legal requirements to debugging and root cause analysis, auditing logs can give developers a history of the system's execution, providing insight into what has gone wrong, when and from where.

Audit logs are usually used for debugging and analysing the system performance in the debugging case. However, mitigation needs to be done if we want to debug an open distributed systems such as the Internet of Things wireless sensor-based networks. With that in mind, researchers have approached this problem and presented debugging solutions for this case.

For example, Liu et al.[135] designed a passive diagnosis solution PAD for wireless sensor networks. With a markup scheme, they included debugging information within the packets, decreasing the communication overhead. As the sink collects the markup packets, it can process these packets and learn how the network behaves, thus making it easier to pinpoint the abnormal activities in the network.

On the other hand, Schneier and Kelsey [171] introduce a secure scheme to store authenticated audit logs. They introduce a model with an untrusted entity and a trusted one. Periodically, the untrusted entity has to send logs to the trusted entity, where each log is encrypted individually with its encryption key along with a hash chain that contains parts of each written log. The hash chain ensures the authenticity of all the previously written logs. The authors indicate the limits of their model by declaring that there is no means to be able to prevent an attacker from editing or deleting any log once he gains control of the entity. However, the model ensures that the attacker cannot edit the past logs, except for deleting them, which is easy to detect because of the hash chain.

Audit logs are also a central part of intrusion detection systems, and these systems use them to capture security breaches by detecting abnormal activities in the system. Lunt [138] demonstrates the use of these audit logs for automated security audit logs analysis. On the other hand, Teng et al.[190] introduce a way to analyse these security audit logs using inductive learning.

In a different research approach, Jiqiang et al.[108] propose the use of audit logs for forensic purposes, where they introduce a secure audit log server that resides on a remote network, and that receives data from a collecting module which resides on the host machine. The host machine logs information about its operations according to a structure predefined by the network administrator. The network collecting module, which resides at the network's entry point, logs information about the network. The server ensures the audit logs' integrity and dependability, making it useful for forensic investigations.

Similarly, the cloud environment uses audit logs for various reasons. The preservation of audit logs in the cloud for forensic purposes, for instance, has been accomplished by Sumalatha and Batsa in [187], where they applied clustering and segregation methods to the Cassandra database. Their model introduces a method to create and analyse audit logs more efficiently. On the other hand, the authors in

[144] use audit logs to ensure the compliance of the data locations in the cloud with the legislation. Another form of cloud computing is clusters, a group of computers connected to work as a single unit. Clusters use audit logs to solve debugging problems, monitor the cluster nodes' behaviour and create recovery points in case a node crashes [123]. Recording network history and events for auditing reasons have been studied extensively in the context of distributed systems in general. However, how to do so in IoTWS networks in the face of Byzantine behaviour has received comparatively less attention.

2.3 Byzantine Fault Tolerance and Fair Exchange

In this research, we examine the issue of reliable logging related to Byzantine behaviours; therefore, we extensively rely on prior findings in Byzantine fault tolerance. In this section, we investigate previous research that relates to our issue.

2.3.1 Byzantine Fault Tolerance

Researchers have been working on Byzantine fault tolerance for decades, which is concerned with developing dependable systems that can cope with various fault modules. The abstract formulation of the Byzantine General problem echoes the challenge of dealing with many behaviours. In general, the Byzantine dilemma is about reaching a consensus on a set of processes, some of which are faulty. Lamport [127] concluded that the number of faults in the system should be less than one-third of the total processes required to solve the problem.

This result has an impact on practically all work in the fault tolerance field, and researchers frequently work around this number of faulty processes by defining a weaker version of the problem or changing the system models [93] [76] [9] [142]. It is also worth mentioning that all of the important work in this subject has been done with system models incompatible with the IoT wireless sensor-based network paradigm, which prompted this research. We discuss the fault tolerance approaches for IoT wireless sensor-based networks next.

It is assumed that the system is synchronous, that is, that there is an upper bound on the delivery of messages, and that the system can identify missing messages and the problem is applied to the process that delivers disinformation. The results of [74] particularly bound the consensus problem in a synchronous system with at least $3f + 1$ accurate processes and $2f + 1$ graph connectivity, where f is the number of faults.

However, solutions for the asynchronous communication model are constrained by the impossibility that consensus in an asynchronous system cannot be reached

deterministically if one process can crash [85]. Approaches like randomisation, failure detectors, and partial synchronisation have been introduced to overcome this impossibility [66].

In contrast, the author in [46] investigated asynchronous communication and demonstrated that a probabilistic protocol could tolerate up to $(n-1)2$ failures for fail-stop failures and that solving the consensus problem with more than $(n-1)2$ failures is impossible. In the case of malicious processes, the $(n-1)/3$ threshold still applies, as does the impossibility of solving the problem with more than $(n-1)3$ failures, with the number of processes in the system being represented by n [46].

In another work, the authors in [35] offer an algorithm that can tolerate $n > 4t$, with t being the mobile Byzantine fault, to solve the consensus problem in the presence of Byzantine mobile faults.

The authors in [120] offer a reliable Byzantine resilient broadcast, but there is no mention of logging in work or whether it could operate in a wireless sensor network. Whilst [146] focuses on attaining reliable communication with Byzantine nodes, the approach provides reliable communication in cryptographic and non-cryptographic scenarios; however, message complexity can be addressed; data collection is not required.

In [153], the authors investigate the topic of topology discovery in the presence of Byzantine nodes and provide a polynomial message complexity approach. The nodes share information about their surroundings until they have figured out the network's topology in its entirety or parts. They describe two algorithms: Detector and Explorer; the detector handles the weaker problem with less message complexity. Although the explorer solves the strong form of the problem, the message complexity is more significant. The presented approach, however, presupposes a trustworthy channel in which messages are delivered in FIFO and not lost or corrupted, which is irrelevant in a WSN scenario.

2.3.2 Fair Exchange

Fair exchange is a problem that originated in the field of electronic commerce in the 90's. The setting for fair exchange is one in which two potential parties must exchange items in an asynchronous distributed system. As such, the problem attempts to achieve an effective state whereby some fairness property is achieved [28]. The solutions for fair exchange fall into two broad categories:

- *Gradual Exchange protocols:* These protocols are based on both parties disclosing the items gradually. It results in a communication overhead, which is not the preferred solution.

- *Trusted Third Party Protocols (TTP) - (Optimistic):* These protocols depend on the notion of a trusted third party to ensure the fairness of the exchange between the two parties and is sometimes the favoured solution [212].

Asokan [26] defines the properties of a fair exchange solution as effectiveness, fairness, and timeliness. These then form the bases of the proposed optimistic fair exchange protocol introduced in [27] with four messages in a synchronous system. Later the authors improved on their result [29] by introducing a protocol version with four messages in an asynchronous model. They made use of encryption to implement a fair exchange of signatures. The authors in [36] also use encryption to introduce fair exchange protocols with offline trusted third parties.

In Asokan's work [26], fairness was thoroughly studied and led to the proposition of two main types of fairness, namely strong fairness and weak fairness, which Pagnia et al. [200] used to build their definition of fairness. They introduced six degrees of fairness for fair exchange problem protocols.

Then, Ezhilchelvan and Shrivastava in [83] proposed a family of fair exchange protocols that presupposed a trusted third party. They categorised these protocols based on assumptions about the users and the systems. For example, they categorised users based on their conduct, which could be restrictive or malignant, and systems based on their communication mode, which could be synchronous or asynchronous. The usage of TTP is a significant aspect in all of these protocols, which are often employed to circumvent the impossible problem of addressing the fair exchange problem in the presence of malicious nodes [157].

Due to the emergence of cryptocurrencies and blockchains in recent years, the problem of fair exchange has received increased attention. Using characteristics of fair exchange, researchers improved their systems to create protocols that incorporate the best of both domains. In [132], researchers investigate the fairness of transactions and introduce the concept of *strong timeliness* to make cryptocurrency transactions fair.

One approach to integrating the two is to use blockchains as a trusted third party in the event of a dispute, akin to the FairSwap [80] and zero-knowledge contingent payment protocols[147]; these protocols enable the execution of smart contracts on a blockchain.

When comparing the use of blockchain versus TTP for ensuring trust, it is essential to note that miners have a role in blockchain, while in TTP, only the parties involved in the item transfer are involved. However, recent research has shown a potential problem with using blockchain this way. An attacker can manipulate the fairness of a transaction through Miner Extractable Value (MEV), allowing those in possession of transferred items to bribe miners and compromise the transaction's

fairness. Thankfully, the authors of [62] have come up with a solution called *Ponyta* to address this issue. This protocol is designed to be resilient and fair, effectively mitigating the problems associated with MEV.

In this study, we are interested in the variant of fair exchange that introduces the notions of *strong fairness* and *weak fairness*.

Chapter 3

System Model

To better understand the problem of the reliable logging collection and the case of auditability, we first explore these problems theoretically. We develop a solution and explore the validity of this solution through several experiments.

In this chapter, we explain the models that are the basis of our theoretical and experimental results. We start by defining the node models; the operating systems used, the routing protocol and the related requirements for setting the node individually. Then we define the general network model in terms of the communication, fault and process model.

3.1 Node Model

The node model is explained in this section. When simulating this work, this involves the operating system, routing protocol, and MAC protocol utilised for each node in the network. These node-level models govern how nodes operate in the network.

3.1.1 Contiki-NG

We used the Contiki-NG [156] operating system in all the simulations and experiments that we have conducted in this work. Contiki-NG is an operating system for the next generation of IoT that has forked from the Contiki OS [2].

It is open-source and cross-platform; it concentrates on secure and reliable low power communication protocols, such as IPv6, RPL and CoAP. Figure 3.1.

Contiki-NG applications are created using a Process abstraction, and processes are built on a threading library called Protothreads [125]. Contiki-NG uses two modes of events: asynchronous and synchronous.

A process will be idle until it receives an event; once an event is received, the

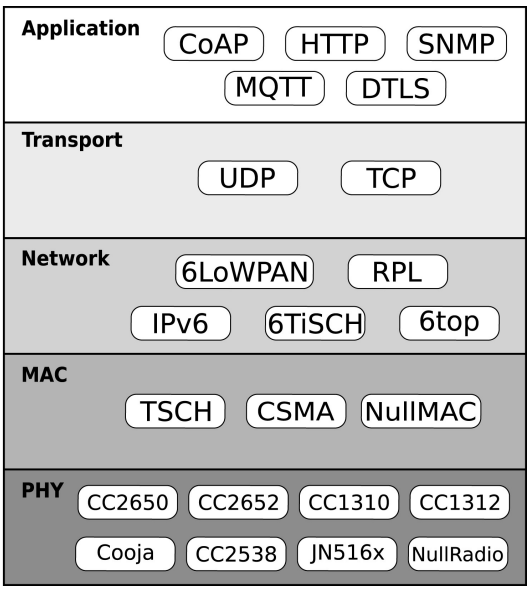


Figure 3.1: Contiki-NG Network Stack [156]

process will perform a chunk of work, and when it is finished, it will suspend the execution until another event is received. An event could be a timer expiration, an incoming packet, user action, sensor-driven or a user-defined event.

Contiki-NG provides a wide range of networking modules and services:

- IPv6 Networking stack and multicast.
- The RPL routing protocol.
- Constrained Application Protocol (CoAP) and Constrained Application Protocol Secure (CoAPs).
- Carrier Sense Multiple Access (CSMA).
- TSCH and 6TiSCH.
- 6TiSCH scheduler Orchestra.
- Communication Security.
- Simple Network Management Protocol (SNMP).

In this work, we used CSMA with the IPv6 networking components, the RPL routing protocol, and TSCH 6TiSCH and Orchestra in the other experiments.

Contiki-NG also supports different platforms, such as:

- Cooja: Cooja native motes platform.

- cc2538dk: TI cc2538 development kit.
- openmote-cc2538: OpenMote cc2538.
- sky: Tmote Sky / TelosB
- zoul: Zolertia Zoul platforms: Firefly, RE-mote, Z1 and Orion.

We use the Sky and Zolertia Z1 platforms along with the Cooja Mote in this work. We give a brief summary of their key features:

- *sky*:
  It used to be the default platform for the Contiki OS [2]; it has an 8MHz Texas Instruments MSP430 microcontroller with $10k$ RAM and $48K$ Flash, with integrated humidity, temperature and light sensors. However, due to RAM and ROM constraints, the Sky can only run a part of the stack [65].
- *Zolertia Z1* :
  This is a platform that is compatible with the tmote family of microcontrollers and features upgrades that double the performance. It has an MSP430F2617 microprocessor with 8 kilobytes of RAM and 92 kilobytes of Flash memory. Z1 memory is capable of running all Contiki-NG network stacks, and its memory capacities allow for diverse application implementations [1].
- *Cooja Mote* :
  This is the virtual platform for the Contiki-NG; it builds and links all hardware access straight to the Cooja simulator [3].

Contiki-NG also implements the COOJA simulator, which allows for simulating virtual wireless sensor networks for testing and debugging purposes. It provides several parameter options to control the experiments and a GUI and NON-GUI running option. Before testing our algorithms in the test-bed, we utilise the Cooja simulator first.

3.1.2 The RPL Routing Protocol

The IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) is the primary routing protocol that we used while performing simulations and experiments for our work. The RPL protocol creates a distance oriented directed acyclic graph *DODAG* tree-like topology that is routed towards a single destination; this destination is usually called the *root*. Each node in the DODAG will have a rank related to the node position in the DODAG, and the rank will be calculated based on network

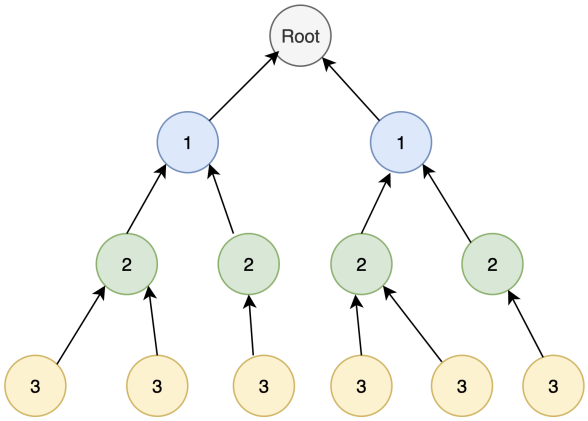


Figure 3.2: RPL DODAG Example.

metrics that are specfied by the *objective function* ***OF***. The rank strictly decreases up to the root and increases down the DODAG.

Depending on the network nature, the ***OF*** will be optimised accordingly. The default ***OF*** in Contiki-ng implementation of RPL utilises the *Expected Number of Transmission* ***ETX*** to build a logical routing topology above the physical network topology. One can also utilise the energy reserve of the node as the metric for the ***OF*** or the link quality as well. This logical topology created by the ***OF*** defines the RPL Instance for one or a set of DODAGs.

RPL has three different types of control messages that are used to establish the DODAG:

- DIO - DODAG Information Object.
- DAO - DODAG Destination Advertisement Object.
- DIS - DODAG Information Solicitation.

The DODAG constructions start with the DODAG root broadcasting the DODAG information through a DODAG Information Object (DIO) message. Once the root neighbours receive this message, they process it, then transport it to other nodes and decide whether to join the DODAG or not. When the nodes decide to join the DODAG, it calculates the path to the DIO sender, and the DODAG root becomes this node parent. The node then calculates its rank within the DODAG and then replies with a destination advertisement object (DAO) to its parent, to announce that it has joined the DODAG.

A node that has not received a DIO and would like to join the DODAG can send a DODAG Information Solicitation (DIS) to obtain a DIO from a RPL

neighbouring node. This process is repeated until all the nodes join the DODAG.

RPL implements its own repair mechanisms to keep the network reliable. One of these is using the Trickle timer [130] to keep the DODAG running while also performing Loop Detection, Loop Avoidance, and DODOAG repair. RPL is able to efficiently maintain network consistency [172] by sending control messages and enforcing restrictions on the nodes, such as preventing a node from choosing a parent with a lower rank and setting a network lifetime [118].

RPL supports upward and downward routing as well as point-to-point, multipoint-to-point, and multipoint-to-multipoint communications. A packet from one node to another first reaches a shared ancestor; if there is no common ancestor, the packet is routed to the root and then to the destination. However, RPL's outstanding performance is a result of its upward routing, whereas its downward routing and point-to-point connections are less effective [211].

Routes become unstable in the downward [79] or point-to-point direction when the destination is not on the route of the sender parent. There are two reasons for this: the first is the MAC layer packet drop and the second is the link asymmetry in RPL.

Contiki-NG provides two implementations of RPL: RPL Classic and RPL Lite. RPL Classis is the continuation of the original implementation of Contiki's RPL. Here we explain this difference in relation to Contiki-NG implementation of RPL.

RPL Classic RPL classic is the full implementation of RPL in Contiki-ng. It provide full support for all operating modes, multiple DODAGs, and several RPL instances. Here we provide a detailed explanation of RPL mechanism and the DODAG formation.

RPL Lite RPL light is a version introduced during the development of Contiki-ng. It introduces a version of RPL that excludes some features from RPL classic, resulting in a smaller memory footprint. In addition, it is the default RPL implementation in Contiki-ng.

RPL lite is a refactored version of the original RPL implementation that contains a subset of the most relevant and flexible RPL classic features.

It only supports a single DAG, a single RPL instance, and non-storing mode.

3.1.2.1 RPL Operating mode

RPL supports two modes of operation: ***Storing*** and ***Non-Storing*** modes for downward routing. The full routing table is stored in each node in storing mode, which significantly impacts memory footprint while eliminating the potential of long source-routed headers [67].

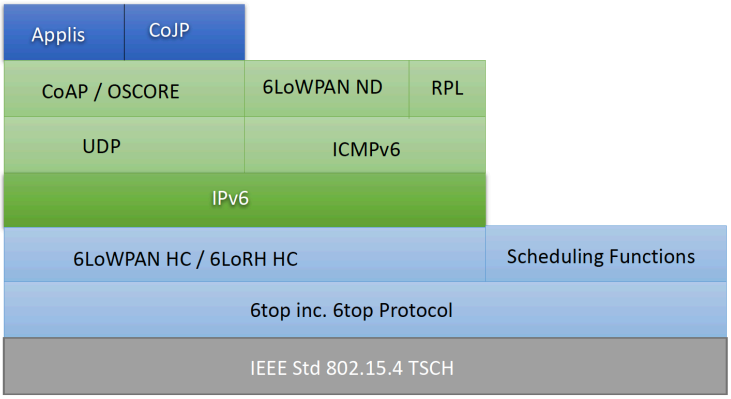


Figure 3.3: Network Stack

On the other hand, in the Non-Storing mode, the nodes do not store the whole routing table for the nodes below them in the DODAG; IPv6 source routing is used instead. Note that in this case, only downward packets get the extra headers.

3.1.3 The Medium Access Control - MAC Model

The MAC, or medium access control, is a critical component of the network stack of a wireless sensor network (see Figure 3.3). It is a part of the data link layer that regulates channel access to various medium sharing devices. Because wireless is a broadcasting medium, when one node sends a message, other neighbouring nodes get it, resulting in collisions and message loss. The MAC protocol can manage message flow, allow for unicast transactions, prevent collisions, and reduce data loss [140]. Most notably, MAC protocols can assist in reducing power consumption by synchronising communication and allowing nodes to go to sleep when not in use, saving a considerable amount of energy.

There are two types of MAC protocols: contention-based and scheduling-based. Contention-based protocols require nodes to wait for the channel to be free before transmitting and waiting for an acknowledgement, resulting in the least amount of processing, making these protocols flexible and scalable to large networks [21].

Each node in schedule-based MAC protocols has its own sending or receiving slots. The node remains sleeping until the time for a previously agreed-upon slot arrives; if it is a sending slot, the node sends a packet; if it is a receiving slot, the node prepares to receive a packet.

We provide a full description of both MAC protocols as they were used in this work's experiment.

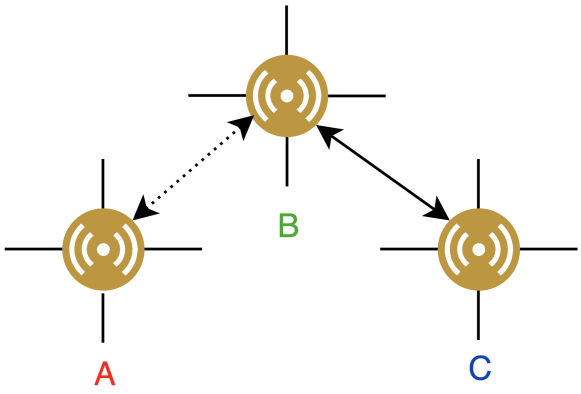


Figure 3.4: Hidden Terminal Problem

3.1.3.1 Contention Based MAC Protocols -CSMA

Contention-based protocols use a single channel for all network communication, which is allocated on demand. This provides numerous benefits to contention-based protocols: One advantage is that because they are allocated on-demand, they scale quickly independently of network density or traffic. Second, they tolerate topology changes quite well, and finally, they do not require any sophisticated synchronisation algorithms. However, one significant downside is that contention-based protocols are inefficient in their consumption of energy [105].

Carrier sense medium access -CSMA is a MAC protocol that is commonly used in wireless sensor networks. The primary idea is to listen before broadcasting in order to detect whether the medium is busy; this is also known as carrier sense. There are a variety of CSMA protocols, including non-persistent, 1-persistent, and p-persistent [41] [88].

In the non-persistent CSMA, when a node detects an idle channel, it sends the message instantly; if the channel is busy, it waits for a random duration before restarting carrier sensing [99]. If the channel is idle in 1-persistent CSMA, the node transmits; otherwise, it listens until the channel is free. However, in p-persistent CSMA, the node sends with a probability p, then backs off with a probability $(1-p)$ and restarts the carrier sense.

When a transmitter alters the direction of its antenna without first verifying the availability of the channel, an issue known as a directional hidden terminal problem [186] arises. This can lead to communication issues when the node attempts to communicate with the receiver without knowing if the channel is free (due to the direction of its antenna). As shown in Figure 3.4, let us assume that node A is sending to node B, but as node B is busy receiving a message from node C (node B antenna is directed towards node C), causing collisions and message loss.

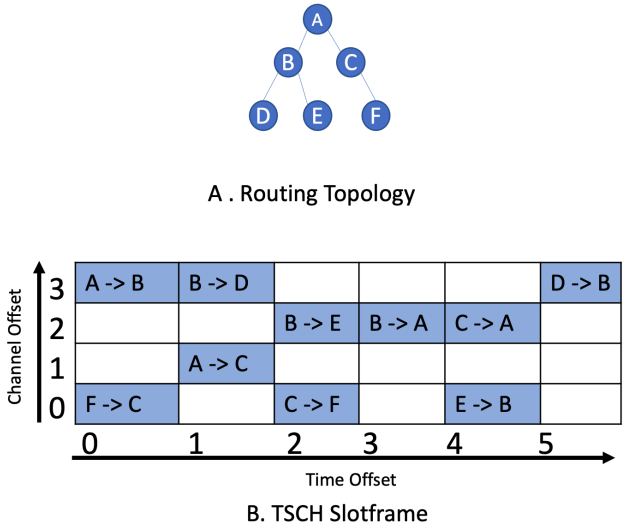


Figure 3.5: (A) A Routing Topology Example. (B) A TSCH Scheduling example, where the time division is represented by the time offset on the X-axis and the frequency division is represented by the channel offset on the Y-axis.[82]

This difficulty resulted in the development of CSMA versions that included collision avoidance measures like the handshake and the use of acknowledgements.

3.1.3.2 Schedule Based MAC Protocols -TDMA/TSCH

One of the most widely used schedule-based MAC protocols is TDMA. It is an abbreviation for *Time Divided Medium Access* [179]. The basic idea is to partition the channel into N time slots, each of which allows only one node to communicate. The time slots construct a frame that is repeated throughout the network's lifespan.

The downside of TDMA is that it needs clustering of nodes, and a cluster head operates as a base station, limiting node communication with the cluster head or inside the network. This also makes TDMA difficult to expand to large networks because it necessitates modifying the time frame and slot numbers to allow any new node to join the network. To allocate slot boundaries, TDMA protocols also rely on distributed time synchronisation algorithms.

The *Time Slotted Channel Hopping - TSCH* [78] protocol is a variant of the TDMA protocol that was inspired by WirelessHART and ISA100.11a. TSCH creates a global mesh network by utilising channel hopping and time slotting.

Time synchronisation flows from the root to the leaf nodes in a directed acyclic network structure. After hearing a beacon from another node, nodes join the network. Every time a node receives a message from its time source parent, it updates its time.

Time is divided into time slots, and time slots are grouped into slot frames that are repeated over time, just as in TDMA. Time slots typically range between 10ms and 15ms long, giving ample time to send a packet and receive an acknowledgement. Within each slot, a scheduler determines whether to transmit, receive or sleep. A time slot is recognised in a slot frame by its time offset (when in the slot frame it will happen) and channel offset (the channel frequency to communicate on). Slots may be dedicated or shared, contention-free or with CSMA back-off.

Each time slot in the TSCH schedule corresponds to a distinct frequency for each slot frame iteration. As a result, packets are transmitted between nodes at varying frequency. When a transmission fails, it is retransmitted on a different frequency.

As was noted in the preceding chapter, TSCH offers significant benefits to WSN, especially in industrial internet of things networks; thus, there has been a significant amount of research into developing schedulers that optimise TSCH for optimal performance. In this study, we employ the 6TiSCH and Orchestra schedulers; the corresponding chapter provides a comprehensive overview of both schedulers.

3.1.4 Fit IOT-LAB

In order to get more detailed and realistic performance measures, throughout this work, we tested our algorithms in a testbed. Specifically, we used the FIT IoT-LAB testbed for our various tests.

FIT IoT-LAB is a cross platforms testbed; it provides over 1500 nodes across six different locations in France, i.e., Grenoble, Lille, Lyon, Nantes, Paris, Saclay and Strasbourg. Each location has its physical topology, which allows for various testing scenarios, including mobility.

FIT IoT-LAB also supplies different communications techniques through introducing several radio technologies to allow testing isolation, coexisting, cooperation and interoperability scenarios. Technologies like IEEE 802.15.4, Sub-GHz, Bluetooth Low Energy and LoRA are available; however, in this work, we make use of the IEEE 802.15.4.

There are also a variety of hardware boards to test on in the testbed. Aside from the FIT IoT-LAB own boards: IoT-LAB M3 and IoT-LAB A8-M3, there are several market-based boards such as Raspberry Pi 3, Zigduino, Zolertia Firefly and Arduino Zero among others. In our experiments on the testbed, we used the IoT-LAB M3 board.

FIT IoT-LAB supports multi-operating systems that include Contiki-NG, RIOT, Zephyr, and MicroPython. As mentioned earlier, we use Contiki-NG for all the simulations and the firmware used for experiments in FIT IoT-LAB.

In particular, we used the physical topologies of Grenoble and Lille sites, with about 150 nodes for each experiment, that were tested for up to 24 hours. In each results section, in which we present results from the FIT IoT-LAB, we provide the specific deployment configuration for that experiment.

3.2 Network Model

This section will present our theoretical system model, the network model that served as the foundation for this research.

3.2.1 Graphs and networks

An IoT node is a computational device that has both constrained computational resources, such as a weak CPU and small memory and other sparse resources such as a low bandwidth radio interface for communication and a finite energy source. We assume that the small memory only allow a finite number of messages to be stored before being processed. If that message is exceeded, messages are overwritten in a FIFO order.

A node is associated with a unique identifier. A wireless IoT sensor network (WISN)[1] consists of a set of IoT nodes that communicate with each other via their respective radio interfaces. A node i can only communicate with another node j if j is at a maximum distance from i, which is known as the *communication range* of i. Communication in such wireless networks is then typically modelled with a circular communication range centred on a node. Within this model, a node is able to exchange data with all devices within its communication range.

When two nodes i and j can communicate directly with each other, we say that a link exists between i and j. If a link exists between a node i and a node j, then i is a *neighbour* of j and vice versa, i.e., we assume links to be bi-directional. As such, the set of nodes that are linked to a node j is called the *neighbourhood* of j and is defined by J, i.e., $J = \{n|(j, n) \in E\}$.

We thus model an IoT WSN network as an undirected graph $G = (V, E)$, where V represents the set of nodes and E represents the set of links between nodes. We assume the existence of a dedicated node in the network, called a sink, which we denote by Δ, that collects data messages coming from the nodes. Also, the number of nodes in G is $|V| = \gamma$.

A *path* is a sequence of vertices such that each vertex leads to the vertex next to it. Given an undirected graph $G = (V, E)$, we say that G is n-connected if every node $j \in V, j \neq n$ has a path to a node n, which we call an n-path. Then, G is connected if $\forall n \in V$, G is n-connected. We say that G is n^l-connected if $\forall j \in V, j \neq n$, j has h disjoint n-paths. Two paths are internally disjoint only if the end nodes of the two paths are the same while all other nodes differ.

[1] We will use the terms IoT networks and wireless sensor networks (WSNs) interchangeably.

3.2.2 Distributed Programs

3.2.2.1 Syntax

The execution of a WSN node is modelled as a process containing non-empty sets of variables and actions. A distributed programme is then a finite set of $|V|$ communicating processes. The graph G represents the communication network topology of the distributed program.

Each variable takes values from a given domain. We denote a variable v of process p by $p.v$. We assume that each process p has a special channel variable ch, denoted by $p.ch$, modelling a FIFO queue of incoming data sent by other nodes. This variable is defined over the set of (possibly infinite) message sequences. Every variable of every process, including the variable ch, has a set of initial values.

The *state* s of a program is an assignment of values to variables from their respective domains. The set of initial states I_p is the set of all possible assignments of initial values to variables of the programme. A state is called *initial* if it is in the set of initial states. The state space of the program is the set of all possible states of the program.

A state at a process p has the form $\langle name \rangle :: \langle guard \rangle \rightarrow \langle command \rangle$. In general, a guard is a state predicate defined over the set of variables of p and possibly of p's neighbours. When the guard of an action evaluates to *true* in a state s, we say the state is enabled in s[2], and the corresponding *command* can be executed, which takes the programme from state s to another state s', by updating variables of process p. We assume that the execution of any action is atomic.

A *command* is a sequence of assignment and branching statements. Also, a *guard* or *command* can contain universal or existential quantifiers of the form: $(\langle quantifier \rangle \langle bound\ variables \rangle : \langle range \rangle : \langle term \rangle)$, where *range* and *term* are Boolean constructs. A special $timeout(timer)$ guard evaluates to true when a timer variable reaches zero, i.e., the timer expires. A $set(timer, value)$ command can be used to initialise the timer variable to a specified value.

3.2.2.2 Semantics

We consider a programme $\mathcal{P}$ to consist of n processes $p_1 \dots p_n$. Each process has a non-empty, finite set of variables and states. The value of a given variable is drawn from a given domain. An assignment of values to variables is called a *state* s and the set of value assignments defines the state space of $\mathcal{P}$, which we denote by $\mathcal{S}$.

We model a distributed program ϕ as a transition system $\phi = (\epsilon, I, \tau)$, where

[2]By extension, we say a process p is enabled in state s if at least one of its actions is enabled in state s.

ϵ is the state space, $I \subseteq \epsilon$ the set of initial states, and $\tau \subseteq \epsilon \times \epsilon$ the set of transitions (or steps).

A computation of ϕ is a maximal sequence of states $s_0 \cdot s_1 \ldots$ such that $s_0 \in I$, and $\forall i > 0, (s_{i-1}, s_i) \in \tau$. An execution (or trace) σ is a finite or infinite sequence of actions $a_1.a_2 \ldots$, where each action is performed by exactly one of the processes. Denoting the set of actions by $\mathcal{A}$, we denote the set of traces by $\mathcal{A}^*$. We say an event e has occurred *iff* $\tau = (s_i, s_{i+1}) \in a$. [3]

A state s is terminal if there is no state s' such that $(s, s') \in \tau$. We denote reachability of state s' from s as $s \rightarrow^* s'$, i.e., there exists a computation that starts from s and contains s'.

In a given state s, several processes may be enabled, and a decision is needed to decide which one(s) execute. A *scheduler* is a predicate over the computations.

In any computation, each step (s, s') is obtained by the fact that a non-empty subset of enabled processes in s atomically execute an action. This subset is chosen according to the scheduler. A scheduler is said to be *central* [73] if it chooses *only one* enabled process to execute an action in any execution step.

A scheduler is said to be *distributed* [50] if it chooses *at least one* enabled process to execute an action in any execution step. In contrast, a synchronous scheduler is a distributed scheduler where *all* enabled processes are chosen to execute an action in a step.

A scheduler may also have some fairness properties [75]. A scheduler is *strongly fair* if every process that is enabled infinitely often is chosen infinitely often to execute an action in a step. A scheduler is *weakly fair* if every continuously enabled process is eventually chosen to execute an action in a step.

3.2.3 Communication Model

We assume the following communication-based actions:

- A $rcv(msg, sender)$ guard is enabled at process n if it is a message at the head of the channel $n.ch$, i.e., $n.ch \neq \langle\rangle$. Executing the corresponding command will cause the message at the head to be dequeued from channel $n.ch$, while msg and $sender$ are bound to the content of the message and the identifier of the sender node, respectively.
- A $BCAST(msg)$ (or $send(msg)$) command causes message msg to be enqueued to the channel $m.ch$ of all processes m that are neighbours of n.
- A $send(msg)$ command causes message msg to be enqueued to the channel $m.ch$ of a given process m that is a neighbour of n.

[3] We will make it clear what semantic we used in our results later.

In each case, we assume that the message buffer is of finite size, i.e., the message buffer can only hold a small number of messages. When the buffer is full and a new message is arriving, the oldest message currently in the buffer will be overwritten by the new one, i.e., we implement a FIFO ordering for message purging. Formally: $m.ch =< a > \bullet m.ch' \wedge |m.ch| = max \wedge rcv(M, n) \implies m.ch = m.ch' \bullet < M >$.

We assume that there is a routing protocol $\mathcal{R}$ that computes a best (e.g., shortest) path between every node and the sink Δ. Given a node $n \in V$, $\mathcal{R}(n)$ returns the best path from n to Δ under $\mathcal{R}$.

3.2.3.1 Asynchronous System

An asynchronous distributed system consists of a set of application processes $V = \{p_1, \ldots, p_n\}$ which communicate using messages passing over bidirectional channels.

We initially consider an asynchronous message-passing IoTWS network in which there are no timing assumptions. Specifically, we neither make assumptions on the time it takes for a message to be delivered nor on relative process speeds.

In the asynchronous system setting, we assume the existence of a discrete global clock, a kind of fictional device that allows ease of presentation. However, and more importantly, processes do not have access to it in an asynchronous system.

We do not assume reliable channels. In most of the work in this thesis we focus on asynchronous systems.

3.2.4 Fault Model

In this work, we assume that nodes can act arbitrarily, i.e., nodes are Byzantine. Actions a Byzantine node can perform include, among others, send-omission, receive-omission, sending wrong messages etc. We also assume that Byzantine node can be passive in a computation.

We present the worst-case scenario of placing the Byzantine nodes in the network, with a maximum of $(n-1)/3$ Byzantine nodes, where n is the number of the nodes within the network [46].

We require that the network is at least $(2k+1)$-connected, where k is the number of Byzantine nodes[74]. We also make use of Menger's theorem which states that, for a k-connected network and every two vertices u and v, there exists at least a k internally nodes disjoint paths connecting u and v in this graph.

In this work we assume the weakly p-fair semantics. A computation of p is weakly fair for a sequence of states $s_0 \cdot s_1 \ldots$ such that $s_0 \in I$ and for each $\forall i > 0, s_i + 1$ results from s_i by executing a single action enabled in s_i. Weak fairness implies that

if a program action a is continuously enabled, a is eventually chosen to be executed, meaning, that Byzantine nodes may not indefinitely prevent the normal programme from executing.

Chapter 4

Problem Formalisation

In this work, we intend to develop a middleware layer for the Internet of Things wireless sensor-based networks that do not interfere with the other network layers.

In this chapter, we define what middleware is and the requirements we have for it. We describe what we mean by *Correct Log* and provide context by referring to the prior research that motivated this study.

4.1 The Logging Middleware

Distributed systems typically encompass a number of separate components that must communicate consistently with one another. Wireless sensor network-based Internet of Things can be viewed as distributed systems, from network stack layers to application-specific Internet of Things; all of these components must communicate with one another in order to provide the user with a reliable execution without interfering with the ongoing processes.

A *middleware* is a software layer that sits between network operational levels and application layers [151], dealing with issues such as heterogeneity, distribution and mobility that are common in distributed systems [103] (See Figure 4.1). We introduce a middleware layer that logs a particular process between the application and transportation layers.

4.2 Middleware Requirements

We are exploring the development of middleware to log the histories of an Internet of Things wireless sensor-based network. Specifically, we are investigating the creation of a middleware that could satisfy these specifications (See Figure 4.2):

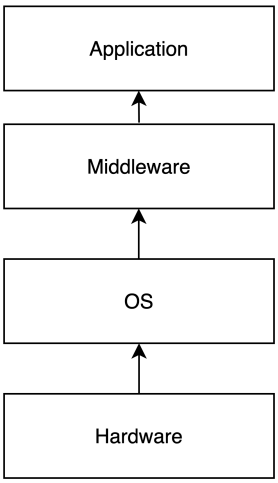


Figure 4.1: Middleware Architecture.

- ***Transparency:*** The logging middleware must not impede the operation of the other network layers.
- ***Fault Tolerance:*** The middleware must be tolerant to network failures.
- ***Scalable:*** The middleware must be capable of managing big networks.

In the transparency specification, our goal was to design a middleware that, when incorporated within the network layers (data, link and physical layer), would not affect the layers' performance. In contrast, in the fault tolerance specification, we required that the middleware would be able to handle faults in the network, in this work, specifically malicious faults. Finally, concerning the scalability specification, the middleware's scalability was required to be able to vary proportionally to the network size.

4.3 The Logging Problem

Before we define what we require from a log to be a correct log, we first define the logging problem informally; a formal definition will follow in the following chapters.

We define an event e in the network as node n sending a data message m with sequence number seq to the sink Δ; we then define β as the set of events that occurred in the network. We define a log entry $l = e(n, m_{seq})$ or $l = f$ where f is a faulty log entry. Then the log file is $L = \bigcup_{i=1}^{n} e_i$.

Thus, we can summarise The logging problem as follows: Can we log all the events in the network?

Figure 4.2: The Logging Middleware Architecture.

Ideally, we would like to be able to log every event that occurred in the network, and in a correct network where there is no malicious behaviour, this is a trivial matter; however, the case changes when we consider Byzantine behaviour, where nodes can masquerade as good nodes by logging fake events or not logging at all.

Additionally, due to the size of IoT networks, distances between nodes span through multiple hops; thus, messages traverse long network distances across unreliable links. One approach to overcome this obstacle would be to reduce the distance over which messages travel, minimising the likelihood of message loss and the consequences of network unreliability; this may be accomplished by assigning several nodes as the logger process and collecting the logs in a distributed manner.

Generally, we assume the existence of an oracle that contains the log of the network execution to allow the verification of the log file. In practice, if we have the required number of nodes related to the Byzantine nodes in the network [74] and the required connectivity, we can verify that the event is within the network.

As previously stated, we need network execution history logs for various goals; this emphasises the importance of formalising and understanding the logging problem, given that once formalised, we can drive implementation at multiple network layers and adjust it in accordance with different application requirements.

The logging problem involves sending a message to a logging entity in order

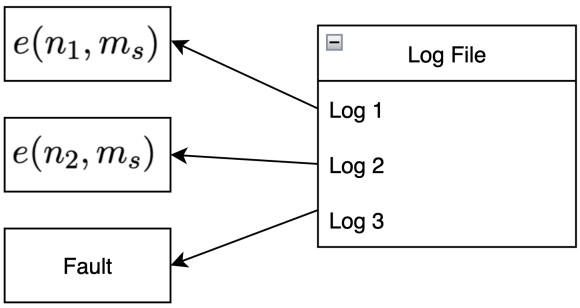


Figure 4.3: Log File.

to preserve a record of the network's execution, but it does not include message sharing. No prior transactions are required for nodes to send logging messages to the logger.

4.4 Correct Logs' Properties

The objective of developing the ***Logging Middleware*** is to guarantee that the logs generated by this middleware are *correct logs*. This section defines what *correct log* means.

Logs in computing are specific files that include information about variables or processes and are generated based on the programmer's setups during the execution. Histories are particular sorts of logs that include records of system events that have occurred in the past. Histories are defined as an auxiliary variable [7] in distributed systems.

In other words, it is a variable that stores the computation history, and its addition to the system does not affect its functions. Thus, throughout this work, the terms histories and logs are used interchangeably, as the correct logs we are constructing this model to collect are the histories of a particular system action. However, in our study, we define logs differently. A correct log in this work is a log, or a history, that is logged by a trustworthy process in the system and has the following properties:

- ***Termination:*** If $e(n, m_s) \in \beta$ then $\exists\ l(n, m_s) \in L : l(n, m_s) = e(n, m_s)$.
- ***Validity:*** If $l(n, m_s) \in L\ \exists e(n, m_s) \in \beta : e(n, m_s) = l(n, m_s)$.

The termination property assures that if an event occurs in the network, a log of that event exists; informally, each data message-sending event in the network will have a corresponding log that records this event. On the other hand, the validity property states that if there is a log entry l_i of an event e_i, then e_i did happen.

These properties can be achieved easily in a connected network with reliable links and correct nodes; however, when we look into the case of Byzantine behaviour, these properties will be harder to satisfy.

The completeness property, for instance, will not be satisfied if the nodes refuse to log the event, and the validity will also be violated if a node maliciously logs a fake event. As we are not using any cryptography primitives to solve this problem, to satisfy these properties to some extent, we need a guarantee that at least a third of the network nodes are correct [74]. In the following chapters, we investigate the problem and show how we can solve the problem of event logging in IoT networks with the presence of Byzantine nodes.

According to our knowledge, any previous formalisation of the logging problem has yet to be conducted, particularly for an IoTWSN environment. However, for the sake of completeness, Nesterenko's [153] work introduced some of the notions of completeness that we employ in our results.

Chapter 5

The Complexity of Reliable Logging

The WSNs are designed to cover vast areas with one base station or a sink, usually positioned in the corner of the fields or the middle. The nodes then send periodic data messages reporting the status of the environment to the base station. Due to the size of the network, packets travel several hops before reaching the sink; occasionally, they get lost during the transfer depending on the network traffic [63]. Furthermore, the links between the nodes are unreliable; node failures disrupt the connection between nodes, e.g. battery expiring, or environmental conditions, e.g. excessive heat, which affects the delivery of the messages [109].

Due to the conditions mentioned above, protocols designed for WSN tend to concentrate on having a high delivery ratio with the minimal consumption possible of power and memory. Usually, achieving this goal means sacrificing the security and trust of the network, making WSNs prone to attacks [205]. As the data collected by the sink is employed to manage its surroundings, it is critical to guarantee the soundness of these data. Ensuring that we have correct, consistent histories of the network events is essential to make sound decisions while investigating the collected data. Whether these decisions are related to debugging, auditing, or forensically examining the network, ensuring the integrity of the histories validates the decision-making verdicts.

For example, let us consider the case of collecting temperature data from a WSN deployed in a hospital. The sensor nodes read the temperature of various hospital departments and report to a base station where different decisions are made accordingly. Other sections of the hospital require different temperature settings. If nodes report incorrect temperature values, fail to report, or due to malicious activity, changes are made to the values; the base station will reach a faulty conclusion,

resulting in catastrophic outcomes.

Ensuring that only correct and consistent data are being analysed is vital. Knowing how the application collected the data through the histories of the network will make it possible to discard faulty data messages or isolate malicious nodes. Consequently, the process of logging the histories of the network needs to be formalised to produce correct logs, ensuring only the correct data are being collected.

As we mentioned in 2.2 and 4, there is no formal definition of the logging problem, and in those works where logs are collected [58] [189] there is always the assumption that either the networks are free of malicious behaviour or that the nodes can be trusted.

In this work, we define the problem of log collection. Since data is collected centrally in IoTWSN, forcing the packets to travel long distances and creating a single point of failure, we propose a distributed logging system by scattering loggers throughout the network, decreasing the distance travelled by the packets and distributing the logs storage. Thus, we split the problem into two main sub-problems:

- The Logger Selection Problem.
- The Collection Problem

In the logger selection problem, we address the issue of unreliable links and the data packets travelling long distances to the sink. Instead of collecting the logs at the sink and creating additional traffic, we propose to collect the logs in a distributed manner, thereby reducing the traffic to the sink and optimising the packet delivery ratio.

The collection problem regulates the log collection process and identifies the specifics to ensure correct logging in the network. We present several variations of the problem, i.e. strong and weak collection. We show it is impossible to solve both versions in an asynchronous system with at least $3f + 1$ correct nodes; f is the number of faulty nodes.

We then introduce a weak probabilistic version to solve the problem up to a probability p. We then present the collector algorithm, which implements the probabilistic solution in the receiver-oriented and sender-oriented versions. We also introduce a centralised version to validate our findings. Finally, we show several simulations and deployment results demonstrating the collector performance in several scenarios.

5.1 Additions to Models

This chapter's work adheres to the network model that was first introduced in 3.2. We then add the subsequent model to that:

5.1.1 Logging System

For logging purposes, we need a system that records (communication) events that are happening in the network and thereby observes the application processes.

For this purpose, we add a set of logger processes to the system. The set of logger processes is denoted by $\mathcal{L}$. We augment the application processes in such a way that when an event e occurs at a process p (denoted by e_p), a control message is sent to all the logger processes by the neighbours of p.

Control messages do not interfere with the application messages sent by the application processes in their algorithms. We assume that control message has all the information required for the logger to identify the sender and the event e_p that the control message refers to. Throughout a computation, logger processes continuously collect control messages and construct a set of observations which can subsequently be used for predicate detection, for testing

In practice, the same hardware that runs the application process can also can run a monitor process.

5.2 Problem Statement

We divide the logging problem into two sub-problems to make it more manageable to solve: (i) The Logger Selection Problem (LSP) and (ii) The Collection Problem. We address the difficulty of selecting several logger nodes to collect the logs in the first problem, and we investigate the problem of log collection in the second. In this section, we start with the logger selection problem.

Using a reliable central mechanism to collect logs raises several issues. First, the messages have to travel a long distance before arriving at the central logging unit, resulting in message loss and compromising the message's integrity. Second, centrally collecting logs is computationally expensive and increases network traffic, naturally increasing message complexity. Indeed, if there are k Byzantine nodes in the network, Dolev [74] states that, to ensure consensus in an arbitrary graph, then $3k + 1$ nodes are required with a $2k + 1$ network connectivity. Thus, it becomes vital to study the viability of decentralised logging in IoTWS networks.

Distributing the log collection by assigning a certain number of nodes as loggers and spreading them throughout the network decreases the distance a message

travels and minimises network traffic and computational power usage.

While solving the problem of LSP, there are several considerations to be aware of when choosing the number of loggers in the network:

- Too few loggers means the messages will still have to travel long distances.
- Too many loggers and the logs will be scattered through the various loggers.

Thus, the key is to pick a number of loggers to be uniformly distributed through the network such that messages will not have to travel too far; this is achieved by setting a bond on the distance messages have to travel.

Definition 1 (Logger Selection Problem (LSP))**.** *Given a network $G(V, E)$, a distance δ, and an integer $\mathcal{M}$, the logger selection problem is as follows:*

Select a set of nodes (i.e., loggers) V' such that

1. $V' \subseteq V$
2. $|V'| \leq \mathcal{M}$
3. $\forall u, u' \in V' \cdot distance(u, u') \geq \delta$
4. $\forall u \notin V', \exists u' \in V' \cdot distance(u, u') < \delta$

Informally, the logger selection problem specifies that (i) there must be a maximum number of loggers, (ii) that two loggers cannot be too close to each other (condition 3) and (iii) that the distance between a node and a logger is bounded (condition 4).

Lemma 5.2.1 (NP membership)**.** *LSP is in NP.*

Proof. To prove this, we need to show the correctness of V' in polynomial time. So, given an instance of LSP as described, and a solution set V', the verification is performed as follows: (i) The first two conditions are trivially verified for V'. (ii) For the third condition, we need to show that the distance between any pair of vertices $(u, u'), u, u' \in V'$ is more than δ. We verify this by first constructing a spanning tree of depth δ, rooted at u, by doing a depth-first traversal on G. Then, for every path from u to any leaf vertex, we need to check if u' is on the path. If this is positive, then V' is not a solution. Else, the process is repeated for every other vertex pair. This verification is achieved in $O(|V|^2)$.

Finally, the last condition is verified in a similar way to the third condition by constructing a spanning tree of depth δ, rooted at a vertex $u \notin V'$. If there exists no path from u to a leaf node that contains a vertex $u' \in V'$, then V' is not a solution to LSP. Else, the process is repeated for all other $u \notin V'$. This is also achieved in $O(|V|^2)$. □

Lemma 5.2.2 (NP hard)**.** *LSP is NP-hard.*

Proof. We first present the Minimum Dominating Set (MDS) problem.

Definition 2 (MDS)**.** *Given a graph $G(U, \hat{E})$, a positive integer K, the MDS problem is to find a set U' such that*

- $|U'| \leq K$
- $\forall u \notin U', \exists u' \in U' \cdot (u, u') \in \hat{E}$

With this definition of MDS, the mapping between MDS and LSP is as follows:

- $U \mapsto V, U' \mapsto V', \hat{E} \mapsto E$
- $\delta \mapsto 2$
- $\mathcal{K} \mapsto \mathcal{M}$

Now, we need to prove that a solution for LSP exists if and only if a solution to MDS exists.

$\Rightarrow$ Let U' be the solution to MDS with a graph $G(U, \hat{E})$, and V' be the solution to the LSP problem for a graph $G(V, E)$, under the mapping defined previously such that $V' = U'$ and with $\delta = 1$. We now need to show that V' is a valid solution for LSP. Since U' is a solution of MDS, $U' \subseteq U$ and $|U'| \leq \mathcal{K}$, then, under the mapping, $V' \subseteq V$ and $|V'| \leq \mathcal{M}$. Also, since U' is a solution to MDS, $\forall u \in U \setminus U', \exists u' \in U' \cdot \text{distance}(u, u') = 1 \leq \delta$, meaning that V' satisfies the third condition also. Finally, consider an edge $(v, v') \in E, v, v' \notin U'$. Now, consider the case where $\exists m, n \in U', (m, v), (n, v') \in E, (m, v'), (n, v) \notin E$. Then, $\text{distance}(m, n) = 3 > \delta$. Thus, under the mapping, V' is a valid solution to LSP.

$\Leftarrow$ Let V' be the solution to LSP with a graph $G(V, E)$ and U' be a solution of the MDS problem for graph $G(U, \hat{E})$. Using the mapping identified previously, we have $U' = V'$, with $\delta = 1$. Now, we show that U' is a valid solution to MDS. Since V' is a solution of LSP, $V' \subseteq V$ and $|V'| \leq \mathcal{M}$, then, under the mapping, $U' \subseteq U$ and $|U'| \leq \mathcal{K}$. Since the distance between a logger node m and a non-logger node n is at most δ, then $\text{distance}(m, n) < \delta = 1$. Finally, as the minimum distance between two logger nodes is at least $\delta + 1$, then U' is a valid solution for MDS.

□

Theorem 5.2.3 (NP-completeness)**.** *The Logger Selection Problem (Definition 1) is NP-complete.*

Proof. This follows from Lemmas 5.2.1 and 5.2.2. □

A possible solution to the LSP problem is to use clustering algorithms, e.g., [70]. Clustering algorithms assign each node in the network to a single cluster head. Similarly, a solution to LSP will need to assign each node to a single logger. More specifically, we are interested in the clusterhead selection. In our work, the set of clusterheads can be the set of loggers and cluster members are nodes reporting to a given logger (i.e., clusterhead)

5.3 The Collection Problem

After identifying a satisfactory solution to the problem of logger selection, we now investigate the problem of log collecting.

In this study, we focus on the problem of recording events generated by IoTWS devices at a designated logger. The events typically relate to data messages that are being sent by the application processes running on IoT devices[1]. When a device n sends a data message to the sink (i.e., when an event is generated by an application process running on n), a (different) device m, called a *witness* device, is expected to confirm this data transmission (i.e., event) by sending a "witness" message to the logger. Such a "witness" message is a control message that confirms that a data message has been sent by n. However, in the presence of Byzantine nodes that can behave arbitrarily, a single witness control message may not be sufficient before such a control message is saved by the logger.

The set of control messages logged at a logger l_i is denoted by CM_{l_i}. An entry in CM_{l_i} has the form $\langle CM, (m, n)\rangle$, which confirms that device n has sent the message m. The set CM_{l_i} therefore captures a (sub)set of data messages, DM_{l_i}, that are claimed to have been sent. We assume that a data message has the form $\langle DM, (m, n)\rangle$. As such, the set DM_{l_i} is defined as $DM_{l_i} = \{\langle DM, (m, n)\rangle | \langle CM, (m, n)\rangle \in CM_{l_i}\}$[2]. We denote by $\mathcal{D}$, the set of data messages that is sent by the network to the sink.

The objective is that for every $\langle DM, (m, n)\rangle$ sent in the network, a certain number of $\langle CM, (m, n)\rangle$ is logged at a correct logger. The number of $\langle CM, (m, n)\rangle$ that is needed is proportionate to the number of faults, or faulty nodes, in the network, which we will come to explain in more detail when we introduce our

[1]We will henceforth use the terms "data message sent" and "event generated" interchangeably, depending on the context.

[2]We will say "DM_{l_i} is the set of data messages logged at logger l_i", though it is actually CM_{l_i} that is the event log.

impossibility results. However, considering our fault and communication models, getting the required number of $\langle CM, (m, n)\rangle$ will not always be the case. Thus, we will introduce two variants of the problem: the *Strong* and the *Weak*.

We start with the strong variant that captures the safety and termination properties of the problem:

Definition 3. *(Strong Collection Problem) A programme is a solution for the strong collection problem if each of the programme's computations satisfies the following properties:*

- ***Safety**: At a correct logger l, DM_l is a subset of the set of data messages that have been sent, i.e., $DM_l \subseteq \mathcal{D}$.*
- ***Strong Termination**: A correct logger l will eventually log a control message $\langle CM, (m, s)\rangle$ for a data message $\langle DM, (m, s)\rangle$ sent, i.e., $\langle DM, (m, s)\rangle \in \mathcal{D} \Rightarrow \exists l, \langle CM, (m, s)\rangle \in CM_l$.*

This definition captures the rigorous variant of the problem; first, the safety property ensures every logged control message in a correct logger has a correspondent data message that has been sent. Second, the strong termination property ensures that a correct logger will always log a relevant control message for every data message sent.

The following definition captures the weak variant of the collection problem:

Definition 4. *(Weak Collection Problem) A program is a solution for the weak collection problem if each of the programme's computations satisfies the following properties:*

- ***Safety**: At a correct logger l, DM_l is a subset of the set of data messages that have been sent, i.e., $DM_l \subseteq \mathcal{D}$.*
- ***Weak Termination**: A correct logger l will eventually log either a control message $\langle CM, (m, s)\rangle$ for a data message $\langle DM, (m, s)\rangle$ sent or a fault message $\langle Fault, (m, s)\rangle$.*
- ***Validity**: A fault message $\langle Fault, (m, s)\rangle$ at a logger l is logged only if there are faulty nodes in the network.*

While the safety property is the same as the strong variant, the weak termination property in the weak variant will ensure that a control message or faulty message is logged for every data message sent. Furthermore, the validity property ensures that a correct logger will only log a fault message if there is a fault in the network., thereby increasing the system's flexibility.

As we will see in the next section, achieving these deterministic solutions in a system with faulty nodes and asynchronous communication is impossible. Therefore, we introduce a weak probabilistic variant to the problem such that the validity property will be satisfied up to a probability p.

Definition 5. *(Probabilistic Weak Collection Problem) A programme is a solution for the probabilistic weak collection problem if each of the programme's computation satisfies the following properties:*

- ***Safety**: At a correct logger l, DM_l is a subset of the set of data messages that have been sent, i.e., $DM_l \subseteq \mathcal{D}$.*
- ***Weak Termination**: A correct logger l will eventually log either a control message $\langle CM, (m, s)\rangle$ for a data message $\langle DM, (m, s)\rangle$ sent or a fault message $\langle Fault, (m, s)\rangle$.*
- ***Probabilistic Validity**: With a probability p, a fault message $\langle Fault, (m, s)\rangle$ at a logger l is logged only if there are faulty processes in the network.*

The probabilistic variant captures the same definition of the safety and the weak termination properties as the weak variant; however, it requires the validity property to be satisfied with some probability p.

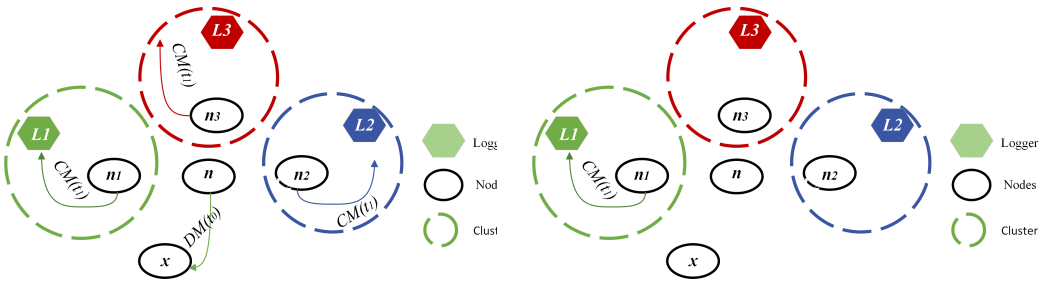
Nonetheless, in several applications, such as auditability and debugging, it is desirable that as many sent messages as possible are logged. The following definitions capture some of the possibilities of interest.

Definition 6. *(Complete Solution) A solution to the collection problem is* complete *if, for every data message sent, there is a corresponding control message that is logged at a correct logger. (Strong Termination)*

Definition 7. *(Partial Solution) A solution to the collection problem is* partial *if, for some data message sent, there is a corresponding control message that is logged at a correct logger.*

Definition 8. *(s-Complete Solution) A solution to the collection problem is s-complete if, for every data message sent by node s, there is a corresponding control message that is logged at a correct logger.*

The complete solution ensures all the data messages sent have a corresponding control message logged in a correct logger. In contrast, the partial solution requires that some of the data messages sent have a corresponding control message. Lastly, the s-complete guarantees that all the data messages sent by s have a correspondent control message.



(a) Computation α_1. (b) Computation α_2

Figure 5.1: Theorem 5.4.1

5.4 Solutions Space

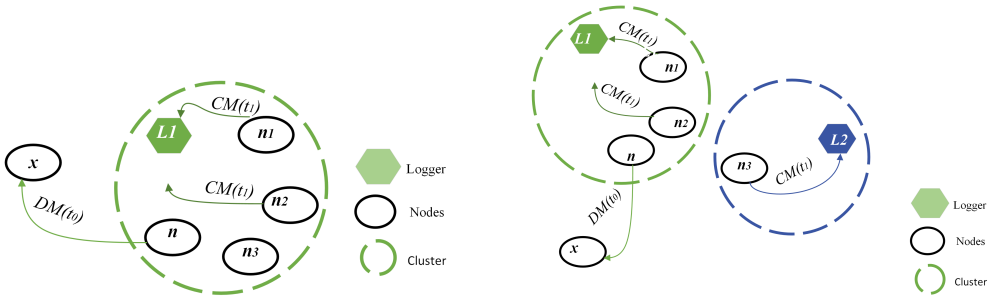
Having defined several variants of the collection problem and possible solutions, we now look at the applicability of these solutions in our defined system. Since the number of Byzantine nodes in the network is k, we further assume that the maximum number of Byzantine nodes in any cluster is $k' < k$. We assume that Byzantine nodes are uniformly distributed across the network, i.e., if there are C clusters in the network, then that each cluster will have a maximum of $\lceil k/C \rceil$ Byzantine nodes, i.e., $k' \leq \lceil k/C \rceil$.

Following clustering solutions for the Logger Selection Problem, we assume the network is divided into clusters $C_1, C_2, \ldots, C_i$. Each cluster C_i has a node member l that acts as the logger. We assume the network is at least $k+1$ connected, and each C is $2k'+1$ connected. Since each C_i is $2k'+1$-connected, each device $n \in C_i$ has a neighbourhood size of at least $2k'+1$ and also has a route to l. Specifically, following the *path* definition in section 3.2.1, for $C_{l_i} \in G$ we assume that G is l_i-connected for all $n \in C_{l_i}$, thus G is L_i^{2f+1} connected..

We now present our first major result.

Theorem 5.4.1. *There does not exist an asynchronous n-complete solution for the strong collection problem for some node $n \in V$ within the graph $G(V, E)$.*

Proof. We prove this by contradiction. We assume that there is a n-complete solution that solves the strong collection problem for $G(V, E)$ with $2k'+1$ connectivity for each C_i in G. We construct three clusters C_1, C_2, C_3, each with logger l_1, l_2, l_3 respectively. We consider a node n that sends a data message $\langle DM \rangle$ to the sink. Each node has a neighbourhood size of $2k'+1$. The topology is as follows: node n has three neighbours n_1, n_2, n_3. Nodes n_1, n_2, n_3 have l_1, l_2, l_3 as loggers respectively.



(a) Computation α_1. (b) Computation α_2

Figure 5.2: Theorem 5.4.2

We construct a fault-free computation α_1 as follows: Node n sends $\langle DM \rangle$ to the sink at time t_0. Each neighbour of n will send a $\langle CM \rangle$ to their respective logger at time t'_0. Assume l_1 receives the $\langle CM \rangle$ at time t_1 but the $\langle CM \rangle$s are delayed for l_2, l_3 until after some time $t_2 > t_1$. At time t_2, since the algorithm is correct, logger l_1 adds the $\langle CM \rangle$ to its log.

Consider a faulty computation α_2 of the solution algorithm, with the same topological information but node n_1 is a Byzantine node, and node n does not send a $\langle DM \rangle$. Now, at time t'_0, node n_1 sends a $\langle CM \rangle$ to l_1, which is identical to that in α_1, which it receives at t_1. At time t_2, loggers l_2, l_3 have no $\langle CM \rangle$ as in α_1 and logger l_1 has a $\langle CM \rangle$ as in α_1. Since a solution exists, l_1 must add the $\langle CM \rangle$ to its log, as in α_1. However, this time no message has been sent by n, violating the safety specification for strong collection. Thus, a solution does not exist, contradicting our assumption that a solution does exist. Figure 5.1. □

The following corollary follows naturally:

Corollary 5.4.1.1. *There exists no complete solution to the strong collection problem.*

Theorem 5.4.2. *There does not exist an n-complete solution for the weak collection problem for some node $n \in V$ within the graph $G(V, E)$.*

Proof. Let us assume that there is a solution that solves the weak collection problem for $G(V, E)$ with $2k' + 1$ connectivity for each C in G. Let us construct two clusters $C_1, C_2 \subset G$, with a neighbourhood size of $2k' + 1$. Let us assume that $k' = 1$ and $n \in C_1$ has n_1,n_2 and n_3 as neighbours, with n_3 as the Byzantine neighbour, and l_1, l_2 are loggers in C_1, C_2, respectively.

We construct a computation α_1 where $n, n_1, n_2, n_3 \in C_1$, n send a data message $\langle DM, (m, s)\rangle$. As n_3 is Byzantine, and it does not send a control message to l_1. At the same time, n_1 and n_2 follow the protocol and send a control message $\langle CM, (m, s)\rangle$ to l_1, l_1, in this case, will receive $k' + 1$ control messages and log a fault message $\langle FM, (m, s)\rangle$ according to the validity property.

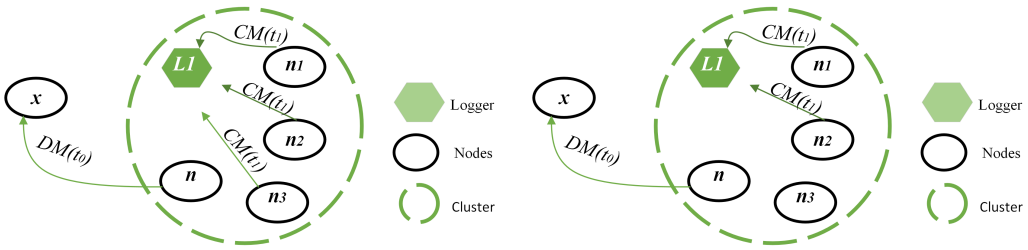
Now consider a follow-up computation α_2 where $n, n_1, n_2 \in C_1$ but $n_3 \in C_2$, and n sending the data message $\langle DM, (m, s')\rangle$. In this computation, n_3 sends a control message $\langle CM, (m, s')\rangle$ to l_2, which is in C_2. Nodes n_1 and n_2 again follow the protocol and send a $\langle CM, (m, s')\rangle$ to l_1, thus l_1 again will receive only $k' + 1$ control messages and a fault message $\langle FM, (m, s')\rangle$ will be logged, since for l_1 the computation α_2 is the same as α_1, which is a contradiction. Figure 5.4.

□

Thus this corollary follows.

Corollary 5.4.2.1. *There does not exist a complete solution for the weak collection problem in G.*

Theorem 5.4.3. *There does not exist an asynchronous n-complete solution for the weak collection problem for some node $n \in V$ within graph $G(V, E)$.*



(a) Computation α_1. (b) Computation α_2

Figure 5.3: Theorem 5.4.3

Proof. We prove this by contradiction. Suppose there is an asynchronous n-complete solution that solves the weak collection problem for $G(V, E)$. Let us construct cluster $C \in G$ with l as the logger, with node n having n_1, n_2 and n_3 as neighbours.

Let us construct fault-free computation α_1, n send $\langle DM, (m, s)\rangle$ to the sink at t_0, each neighbour follows by sending $\langle CM, (m, s)\rangle$ to l at t'_0. l receives $\langle CM, (m, s)\rangle$ from n_1 at t'_1, but the $\langle CM\rangle$ from n_2, n_3 is delayed until $t'_2 > t'_1$. Due to the termination property, l terminates by logging fault message $\langle FM, (m, s)\rangle$.

Now, consider the faulty computation α_2, with n_3 as a Byzantine node, n send a $\langle DM, (m, s')\rangle$ to the sink at t_0, and n_1, n_2 follow by sending $\langle CM, (m, s')\rangle$ to l. However, n_3 does not send any control messages maliciously, and the $\langle CM\rangle$ from n_2 is delayed until $t'_2 > t'_1$. l logs a fault message $\langle FM, (m, s')\rangle$ due to the validity property; resulting in a contradiction as $\alpha_1 \neq \alpha_2$. See Figure 5.3.

Figure 5.4 shows a timeline representation of the proof above. We can observe the timeline of the fault-free computation α_1, where one of the control messages gets delayed and results in a fault log based on the termination property. The second line represents the faulty computation α_2 where malicious behaviour results in a missing control message that will result in a fault log similar to α_1, which is a contradiction. □

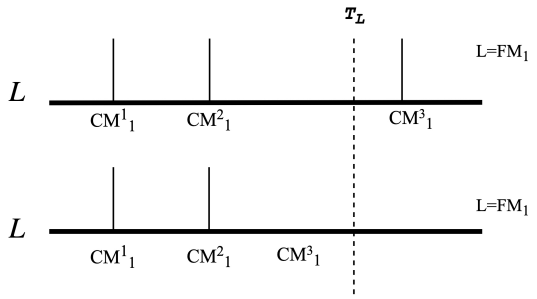


Figure 5.4: A timeline of the computation of Theorem 5.4.3, Figure 5.3 where in the first non-faulty computation, the logger logs a fault due to the delay of the message whilst in the second computation, the logger logs a fault due to malicious behaviour.

Since there does not exist solution to the weak version, we study the weaker probabilistic alternative (Definition 5). Given the problem associated with neighbours belonging to different clusters (i.e., sending control messages to different loggers), we seek a bound on such probability.

Theorem 5.4.4 (Probabilistic Weak Validity)**.** *Given a network $G(V, E)$ with a number of clusters, each node $n \in V$ having a neighbourhood of size N, at most k' Byzantine nodes in each cluster with $n > k'$. Then, weak collection can be solved with probability $1 - (\sum_{i=0}^{k} \binom{n}{k} p^i (1-p)^{n-i})$, where p is the probability of a neighbour node of n having the same logger as the data message sender.*

Proof. Denote the data message sender by s and denote by p the probability of a neighbour node of s having the same logger as s. Let U be the random variable

"number of nodes with the same logger as s". Assume that the size of the neighbourhood of s to be $n > k'$. Then, V follows a binomial distribution with parameters (n, p), i.e., $V \sim Bin(n, p)$.

$$
\begin{aligned}
& \Pr\{\text{no violation of validity}\} \\
= \; & \Pr\{U \geq k+1\} \\
= \; & 1 - \Pr\{U < k+1\} \\
= \; & 1 - (\Pr\{U = 0\} = \Pr\{U = 1\} + \ldots + \Pr\{U = k\}) \\
= \; & 1 - (\textstyle\sum_{i=0}^{k} \binom{n}{k} p^i (1-p)^{n-i})
\end{aligned}
$$

□

Typically, p can be estimated from field data or large-scale experiments.

5.5 Collector

This section presents two versions of the *Collector* algorithm that solves the weak probabilistic collection. One is *Sender Oriented*, where the sender informs its neighbours about its logger, effectively setting the parameter p to 1. The second algorithm is a *Receiver Oriented* one, where the receiving neighbours send control messages to their loggers, setting the value of p to less than 1. For completeness purposes, we also present a centralised version of the *Collector* algorithm, where the sender neighbours send their control messages to the sink, setting $p = 1$

5.5.1 Receiver Oriented Algorithm - $p < 1$

Here, the algorithm works as follows: When a device i sends a message to the sink, each of its neighbours will send a related control message to its logger. When the control message reaches the logger, the logger computes the number of confirmations it has received regarding the sent data message. It logs the control message if the number exceeds a threshold (i.e., exceeds the number of Byzantine nodes in the cluster); see Algorithm 1.

5.5.2 Sender Oriented Algorithm - $p = 1$

Here, the algorithm works as follows, which is almost similar to the *receiver oriented* version: When a device i sends a message to the sink, each of its neighbours sends a related control message to the logger of the sender of the data message. In this case, all neighbours have sent to exactly the same logger, i.e., $p = 1$. When the

Algorithm 1 ***Receiver Oriented - Process*** i

Variables

buf: seq of messages;
detect: Boolean init false;
Δ: Sink;
s: Int init 1;
L: ID; % id of process i's logger
I: Set of nodes; % $\boldsymbol{i}$ neighbours.
UL: Set of unconfirmed logs init $\emptyset$; % if i is a logger.
CL: Set of confirmed logs init $\emptyset$; % if i is a logger, init $\emptyset$.

head(buf) $= \langle m \rangle \rightarrow$ $send\langle DM, (m, s, i), \Delta\rangle$; % send a data message to the sink
$BCAST\langle CM, (m, s, i), \perp, \perp\rangle$;
$s := s + 1$;

rcv $\langle DM, (m', s', j), \Delta\rangle \rightarrow$ %receiving a data message
If $(i \in J)$
$send\ \langle CM, (m', s', j), L, i\rangle$;
End if

rcv $\langle CM, (m', s', j), \perp, \perp\rangle \rightarrow$ %receiving a control message 1st time
If $(i \in J)$
$send\ \langle CM, (m', s', j), L, i\rangle$;
End If

rcv $\langle CM, (nm, ns, k), i, n\rangle \rightarrow$ % i as logger receives CM
If $(\exists\ \langle (m'', s'', k), n\rangle \in UL : (m'' = nm \wedge s'' \neq ns) \vee (m'' \neq nm \wedge s'' = ns))$
$detect := 1$; % messages seem mismatched
End If
If $(\forall \langle (m'', s'', k), z\rangle \in UL : m'' \neq nm \wedge s'' \neq ns)$
$UL := UL \cup \langle (nm, ns, k), n\rangle$;
End If
If $|\{r | \langle (nm, ns, k), r\rangle \in UL\}| \geq f + 1$
$L := L \cup \{(nm, ns, k)\}$; % add to logger
$UL := UL \setminus \{\langle (nm, ns, k), p\rangle | (nm, ns, k) \in L\}$; % remove
End If

Algorithm 2 ***Sender Oriented - Process*** i

Variables

buf: seq of messages;
detect: Boolean init false;
Δ: Sink;
s: Int init 1;
L: ID; % id of process i's logger
I: Set of nodes; % $\boldsymbol{i}$ neighbours.
UL: Set of unconfirmed logs init $\emptyset$; % if i is a logger.
CL: Set of confirmed logs init $\emptyset$; % if i is a logger, init $\emptyset$.

head(buf) $= \langle m \rangle \rightarrow$ $send\langle DM, (m, s, i), \Delta\rangle$; % send a data message to the sink
$BCAST\langle CM, (m, s, i), L, \perp\rangle$;
$s := s + 1$;

rcv $\langle DM, (m', s', j), \Delta\rangle \rightarrow$ %receiving a data message
If $(i \in J)$
$send\ \langle CM, (m', s', j), L, j\rangle$;
End if

rcv $\langle CM, (m', s', j), L\rangle \rightarrow$ %receiving a control message
If $(i \in J)$
$send\ \langle CM, (m', s', j), L', j\rangle$;
End If

rcv $\langle CM, (nm, ns, k), i, n\rangle \rightarrow$ % i as logger receives CM
If $(\exists\ \langle (m'', s'', k), n\rangle \in UL : (m'' = nm \wedge s'' \neq ns) \vee (m'' \neq nm \wedge s'' = ns))$
$detect := 1$; % messages seem mismatched
End If
If $(\forall \langle (m'', s'', k), z\rangle \in UL : m'' \neq nm \wedge s'' \neq ns)$
$UL := UL \cup \langle (nm, ns, k), n\rangle$;
End If
If $|\{r | \langle (nm, ns, k), r\rangle \in UL\}| \geq f + 1$
$L := L \cup \{(nm, ns, k)\}$; % add to logger
$UL := UL \setminus \{\langle (nm, ns, k), p\rangle | (nm, ns, k) \in L\}$; % remove
End If

control message reaches the logger, the logger computes the number of confirmations it has received regarding the sent data message. It logs the control message if the number exceeds a threshold (i.e., exceeds the number of Byzantine nodes in the cluster); see Algorithm 2.

5.5.3 Centre Oriented Algorithm - $p = 1$

In this section, we look at a similar version of the algorithm with $p = 1$ as in the *sender oriented algorithm.*, except that in this version, we set the sink $\triangle$ to act as the central and the only logger in the network. Thus, when a node sends a data message, all its neighbours will send a control message to the sink; see Algorithm 3.

Algorithm 3 ***Centre Oriented - Process*** i

Variables

buf: seq of messages;
detect: Boolean init false;
Δ: Sink;
s: Int init 1;
I: Set of nodes; % $\boldsymbol{i}$ neighbours.
UL: Set of unconfirmed logs init $\emptyset$.
CL: Set of confirmed logs init $\emptyset$.

head(buf) $= \langle m \rangle \rightarrow$ $send\langle DM, (m, s, i), \Delta\rangle$; % send a data message to the sink
$BCAST\langle CM, (m, s, i)\rangle$;
$s := s + 1$;

rcv $\langle DM, (m', s', j), \Delta\rangle \rightarrow$ %receiving a data message
If $(i \in J)$
send $\langle CM, (m', s', j), \Delta\rangle$;
End if

rcv $\langle CM, (m', s', j)\rangle \rightarrow$ %receiving a control message
If $(i \in J)$
send $\langle CM, (m', s', j), \Delta\rangle$;
End If

rcv $\langle CM, (nm, ns, k), \Delta, n\rangle \rightarrow$
If $(\exists\ \langle (m'', s'', k), n\rangle \in UL : (m'' = nm \wedge s'' \neq ns) \vee (m'' \neq nm \wedge s'' = ns))$
detect $:= 1$; % messages seem mismatched
End If
If $(\forall \langle (m'', s'', k), z\rangle \in UL : m'' \neq nm \wedge s'' \neq ns)$
$UL := UL \cup \langle (nm, ns, k), n\rangle$;
End If
If $|\{r | \langle (nm, ns, k), r\rangle \in UL\}| \geq f + 1$
$L := L \cup \{(nm, ns, k)\}$; % add to logger
$UL := UL \setminus \{\langle (nm, ns, k), p\rangle | (nm, ns, k) \in L\}$; % remove
End If

Parameter	Value
Topology	Grid
Network Size	25 - 100
Routing	RPL
Mote Type	Z1
Simulation Duration	24h Cooja Time

Table 5.1: Contiki-NG Cooja Simulation Parameters.

5.6 Evaluation

We now present the viability of our work by presenting the results of several simulations and real-life deployments within a CSMA MAC-based configuration. We first present the simulations, then show the deployments of the protocols in a test-bed and evaluate the results.

5.6.1 Simulations Settings

We used Cooja, a simulator within the Contiki-ng operating system, to carry out the simulations. Contiki-ng is an operating system for resource-constrained devices, such as IoT devices. The default routing protocol in Contiki-NG is RPL, a routing protocol for lossy and low-power networks.

RPL builds and maintains a Destination-Oriented Directed Acyclic Graph (DODAG) topology from a chosen RPL root node. Contiki-ng supports two implementations of the RPL protocol: RPL Classic and RPL Lite. RPL Classic includes the full implementation of the RPL protocol with full functionality, including storing mode. RPL Lite, on the other hand, focuses on the most important and stable functionalities of the RPL Classis, reducing ROM usage. In these simulations, we used RPL lite and the non-storing mode of operation of RPL, as it has a small footprint in the node's memory.

The simulation is started by establishing the network routing using RPL, which means that the DODAG will be formed, and each node will have a routing table for the nodes above it in the DODAG. After the routing is established, we implement the logging layer, including the formation of the clusters. 5.5.

Each logger will broadcast its existence, and the nodes within its range will receive a broadcast and join the cluster. After joining the cluster, each node will send a control message to the logger when one of its neighbours sends a data message. The logger receiving the control message will differ depending on the algorithm used (sender or receiver oriented).

In the sender-oriented algorithm, the control message is sent to the logger

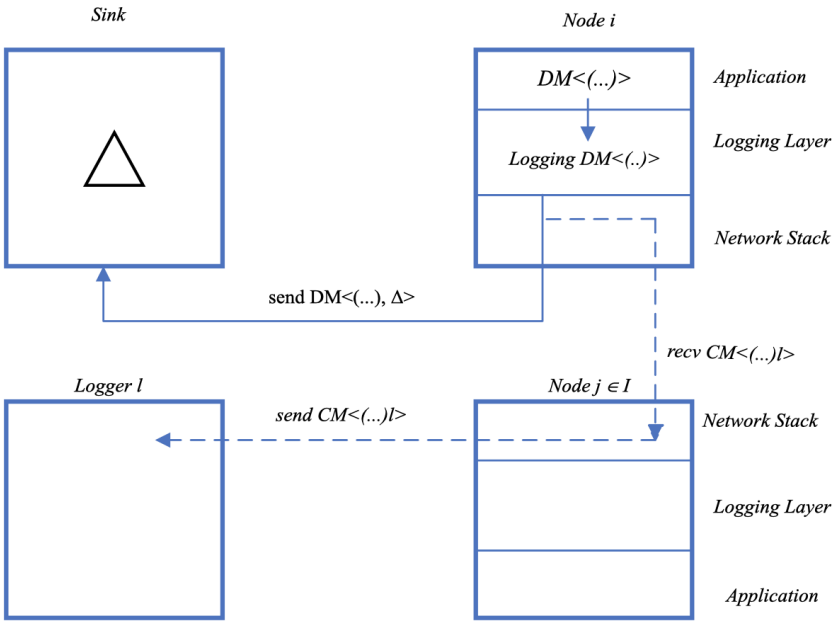


Figure 5.5: The Application Setup.

of the data message sender node. In the receiver-oriented algorithm, the control message is sent to the logger of the node that heard the data message. While in the centre-oriented, all the nodes send their control messages to the sink, which acts as the centralised logger for the whole network.

5.6.2 Simulation Results

We simulated both algorithms in different network sizes and cluster numbers to test the scalability of both algorithms. We simulated four network sizes, up to 150 nodes, in a grid topology. We also reproduced the behaviour of a faulty node in each simulation. Figure 5.6.

5.6.2.1 Receiver Oriented Protocol

As observed previously, when a neighbouring node to a sender has a logger different to the sender, problems may arise such that weak collection becomes impossible. As such, we proved that this can be solved probabilistically. Specifically, we observe that the probability of no violation increases as the value p increases.

Figure 5.7 shows the performance of the receiver-oriented version, i.e., the probabilistic weak collection algorithm. In this version, we can observe the following:

- The high number of logged messages for the data messages sent.

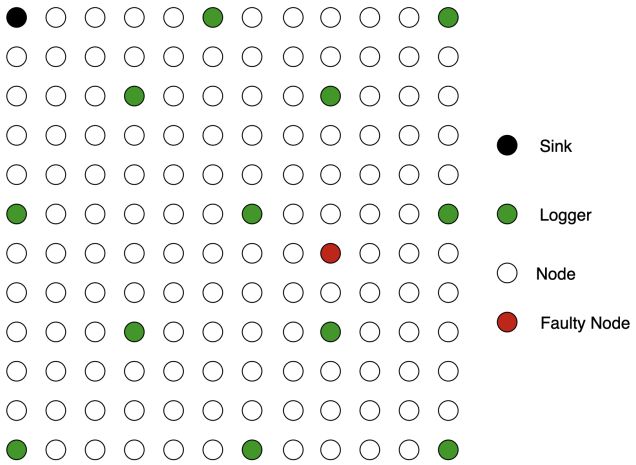


Figure 5.6: An Example of the Grid Topology Used in One of the Experiments.

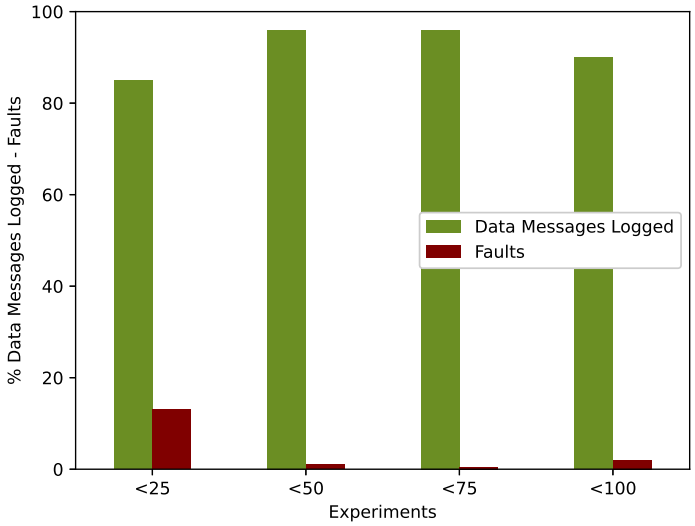


Figure 5.7: The Receiver-Oriented Performance.

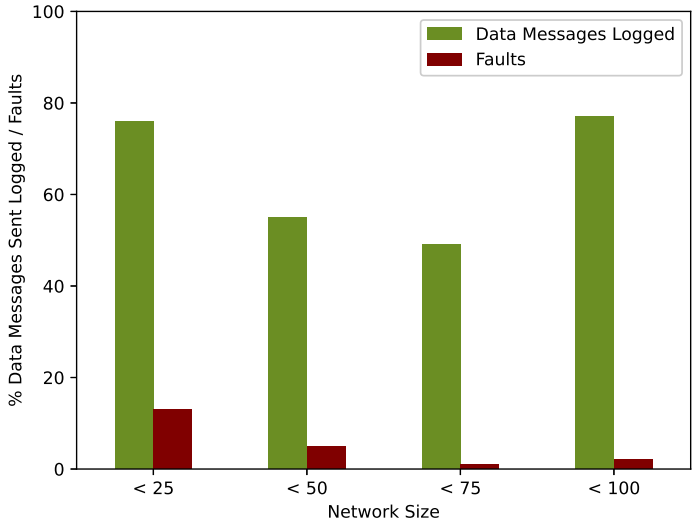


Figure 5.8: The Sender-Oriented Performance.

- The very negligible number of fault messages that are logged, even with a faulty node in the network. As conjectured, with the increasing size of the network, the value of p increases, thereby increasing the number of messages that are correctly logged.

Thus, the receiver-oriented protocol shows that the probability of validity violation can be bounded.

5.6.2.2 Sender Oriented Protocol

In this section, we study a slightly different scenario, where all neighbours are clustered under the same logger (or, for example, the sender notifies who its logger is). As such, we can expect that when there is no fault in the network, such a solution will indeed work.

However, Figure 5.8 depicts a slightly different picture. Specifically, we observe that, counter-intuitively, the percentage of logged messages is less than that of the receiver-oriented. This is because the increasing traffic towards a single designated logger causes a lot of message collisions, resulting in message losses, even after a number of retransmissions.

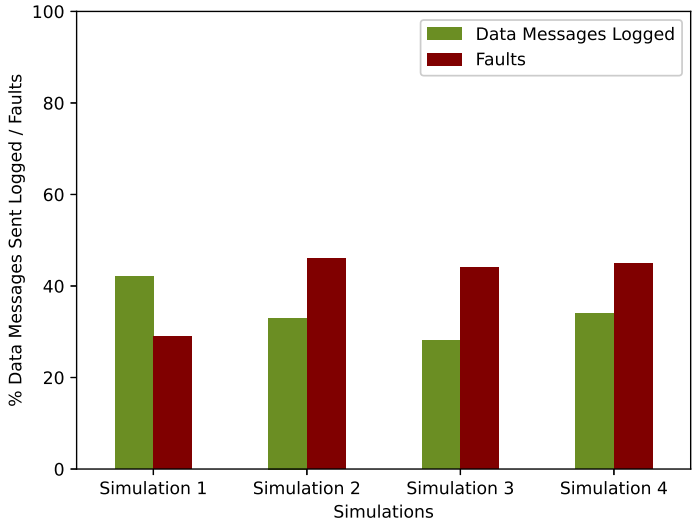


Figure 5.9: The Centre-Oriented Performance.

5.6.2.3 Centre Oriented Protocol

In this section, we evaluate the results of simulating the centralised version we mentioned earlier, where p is equal to 1, as all the nodes send the control messages to the sink. We set the simulations such that all the nodes send their control messages to the sink while placing several faulty nodes throughout the network following the bounds we mentioned in our theoretical results.

Figure 5.9 show the performance of the centre-oriented protocol through several simulations. As the probability of no validity violation increases to 1, we would expect high volumes of correctly logged messages; however, the results from Figure 5.9 tell a different story that can be summarised as:

- The number of data messages logged is less than half of the messages sent.
- The significant amount of faults logged sometimes exceeds the number of data messages logged.

The reason for these numbers is the traffic towards one logger responsible for both receiving data messages and control messages, which increases the traffic in the network, resulting in significant message loss even though the messages are re-transmitted several times.

We, therefore, compare both solutions, receiver and sender oriented, in Figure 5.11. We find that the receiver-oriented algorithm performs significantly better than the sender-oriented approach under high traffic conditions, indicating that the

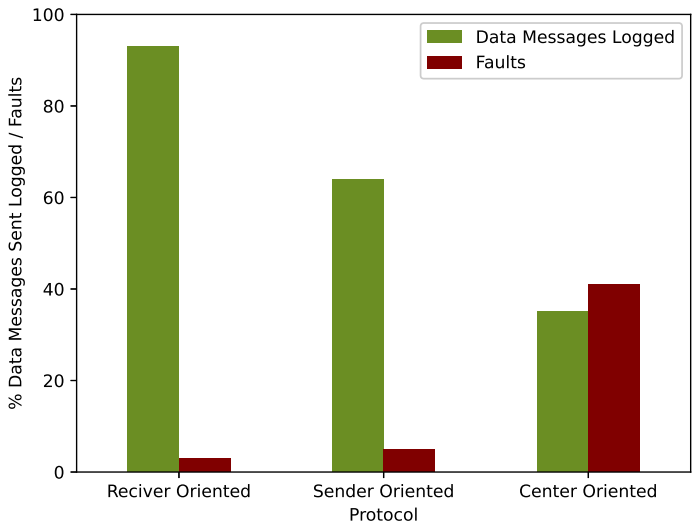


Figure 5.10: A Comparison Between the Performance of the Sender, Receiver and Centre Oriented.

Parameter	Value
Topology	Grenoble & Lille Sites in FIT-IoT lab
Network Size	100
Routing	RPL
Mote Type	M3
44 Simulation Duration	12-24h Real Time

Table 5.2: FIT-IoT Labs Experiments Parameters.

difficulty in selecting different loggers pales in comparison to the impact of decreased link reliability, which in some instances results in discrepancies between the total number of data messages and fault messages logged; this is because of message loss and collisions on a busy network.

When comparing the three scenarios, Figure 5.10, we notice that while the receiver-oriented still outperforms the other two, the centre-oriented falls significantly short in terms of the number of data messages logged, and the increased number of faults logged in the network, thus confirming that the link reliability issue is more crucial than selecting one logger.

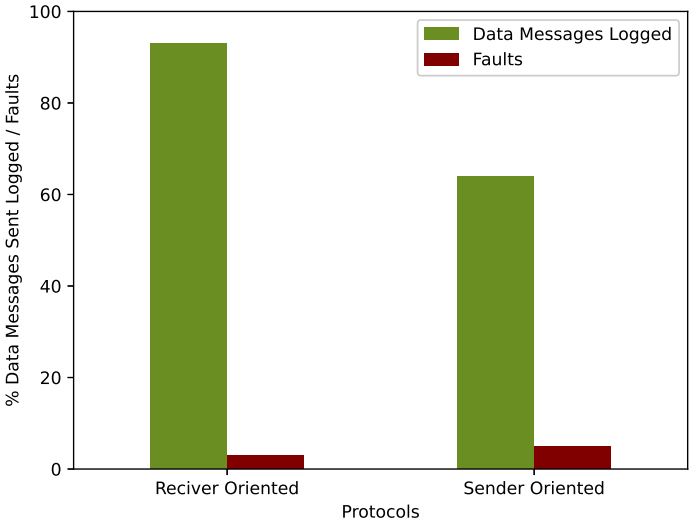


Figure 5.11: A Comparison Between the Performance of the Sender and Receiver Oriented Protocols.

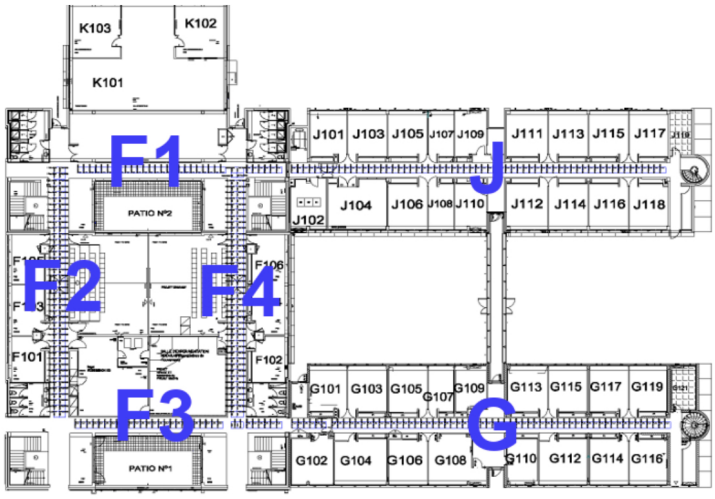


Figure 5.12: FIT IoT-Lab Grenoble Site Topology [13]

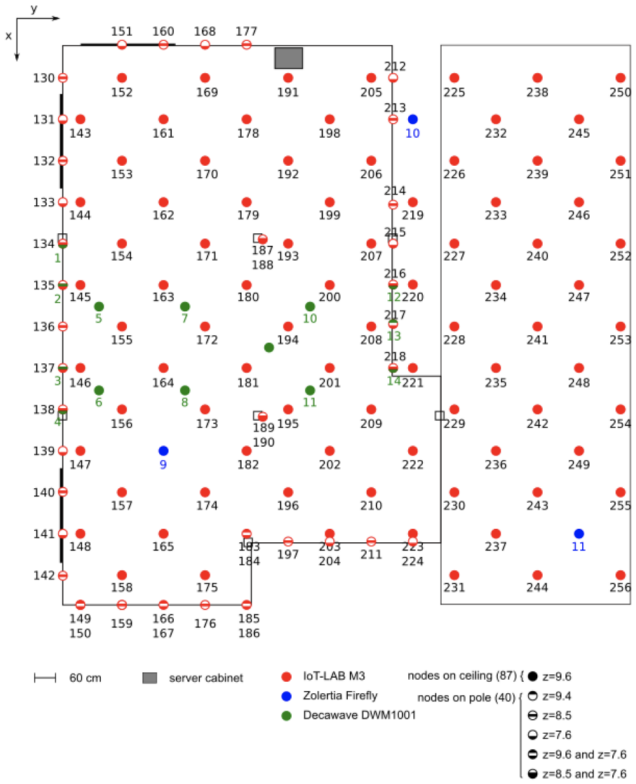


Figure 5.13: FIT IoT-Lab Lille Site Topology [13]

5.6.3 Experiments Settings

To further investigate the efficiency of the proposed protocols, we deployed all versions in a test-bed. Spanning 6 locations throughout France, FIT IoT-LAB [13] accommodates over 2000 heterogeneous IoT wireless sensor nodes, with over 100 mobile nodes equipped with low-rate wireless personal network area (LR WPAN). FIT IoT-LAB provides a web portal and command-line tools for deployment with three types of node architectures: WSN430, M3, and A8.

In our deployment, we used M3 nodes, which feature sensors, a 32-bit CPU, 64 KB of RAM, 256 KB of ROM, and an IEEE 802.15.4-compliant radio interface. M3 nodes are compatible with various IoT operating systems, including Contiki-NG. We conducted a 24-hour experiment in Grenoble and Lille utilising about 115 M3 nodes. With a Contiki-NG firmware that utilises the RPL routing protocol as routing protocol.

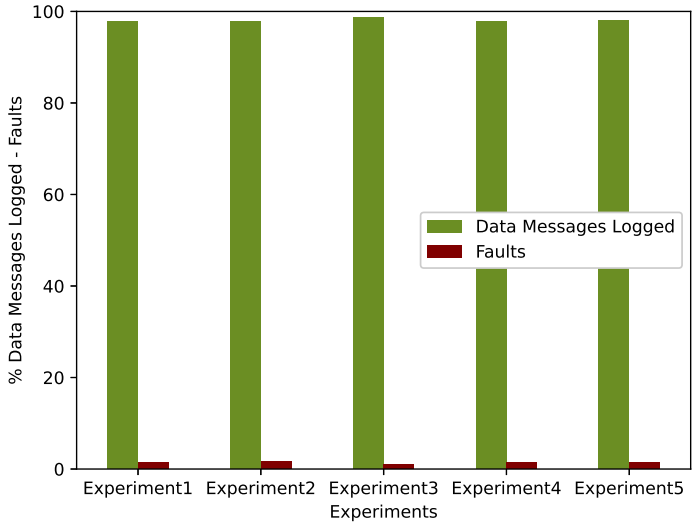


Figure 5.14: The Receiver-Oriented Performance on the FIT IoT-Labs.

5.6.4 Experiments Results

We will investigate the outcomes of the FIT IoT-LAB implementation in the next section. We begin by analysing each version independently, and then present a comparison of the results.

5.6.4.1 The Receiver Oriented Protocol.

Again the receiver-oriented protocol shows the best probability of no violation in the network, regardless of the duration or number of faulty nodes in the network, with each node sending a control message to its logger when they hear a neighbouring node sending a data message. From Figure 5.14, we can summarise the following:

- Almost all the data messages have been correctly logged.
- The negligible percentage of faults in the network.

5.6.4.2 The Sender Oriented Protocol.

In this protocol, each node sends a control message to the data message sender logger, which increases the traffic on one logger and results in message loss regardless of the number of re-transactions, thus increasing the probability of network violations.

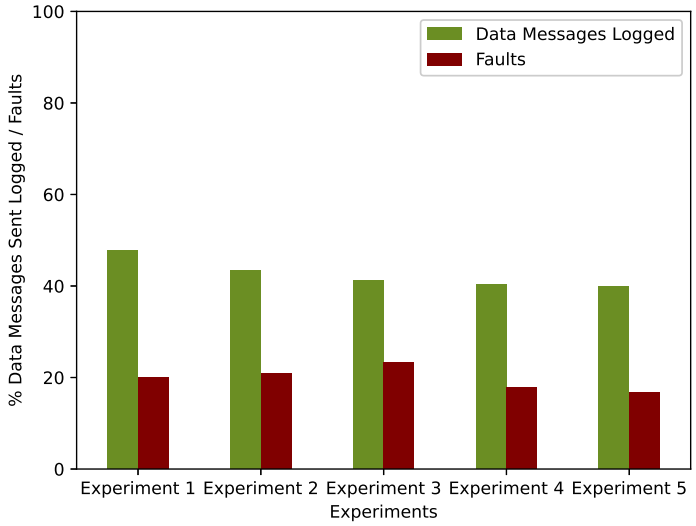


Figure 5.15: The Sender-Oriented Performance on the FIT IoT-Labs.

Figure 5.15 confirms the findings of the previous simulation results. The sender-oriented protocol performance is less optimal than the simulations' performance.

- The number of logged data messages is significantly less than the simulation results.
- The number of faults, in consequence, increases.

5.6.4.3 The Centre Oriented Protocol.

The centre-oriented protocol performs much better on the test-bed than on the Cooja simulator. We can observe from Figure 5.16 the following:

- The increased number of data messages logged.
- The number of fault messages, while less than the simulation results, is still considerably higher than expected.

In Figure 5.17, we compare the performance of both versions of the collector algorithms. We see similar results on the test-bed to what we observed from our simulations. The receiver-oriented protocol still significantly outperforms the sender-oriented protocol, with almost all the data messages sent in the network being correctly logged and a negligible amount of faults being recorded, while the comparison with Figure 5.18 tells a different story than the simulations results in Cooja.

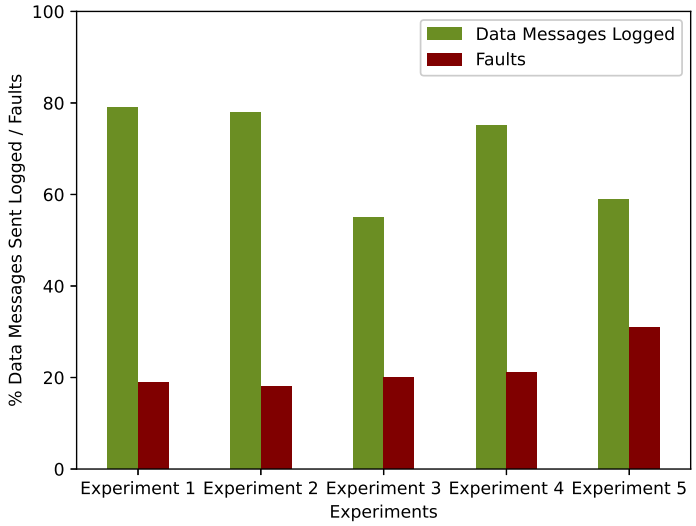


Figure 5.16: The Centre-Oriented Performance on the FIT IoT-Labs.

Though the receiver-oriented still outperforms all the protocols in both simulations and test-bed results, the centre-oriented protocol shows better results on the test-bed than on Cooja, with more data messages logged, due to the powerful performance of the upward routing of RPL.

5.6.4.4 Energy Consumption

We examined the energy consumed by various nodes in a wireless IoT network to assess the merits of our proposed approach properly. As was previously established, IoTWS nodes have minimal resources for generating and dissipating energy.

Accordingly, we analysed the energy usage of every node in the network. We displayed the typical power usage of each algorithm implementation. We calculated the overall mean power consumption of a certain algorithm's nodes and then evaluated how that number stacks up against other algorithms.

Throughout the deployment period in the FIT-IoT labs, we monitored the node's power consumption in watts. The overall energy consumption of the nodes during the *receiver oriented* and the *sender oriented* is represented in Figure 5.19, from this we can conclude the following:

- The *receiver oriented* energy consumption mean is almost the same over time.
- The *sender oriented* energy consumption is unstable, with several considerably higher energy spikes occurring several times during the lifetime of the

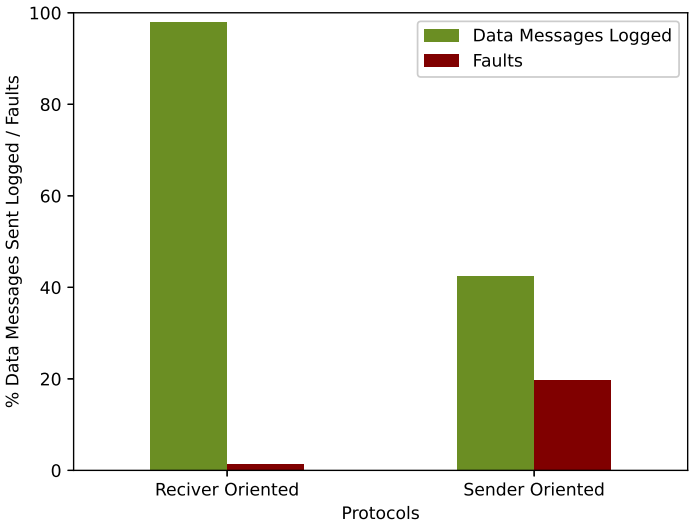


Figure 5.17: A Comparison Between the Performance of the Sender and Receiver Oriented Protocols on FIT IoT-Labs.

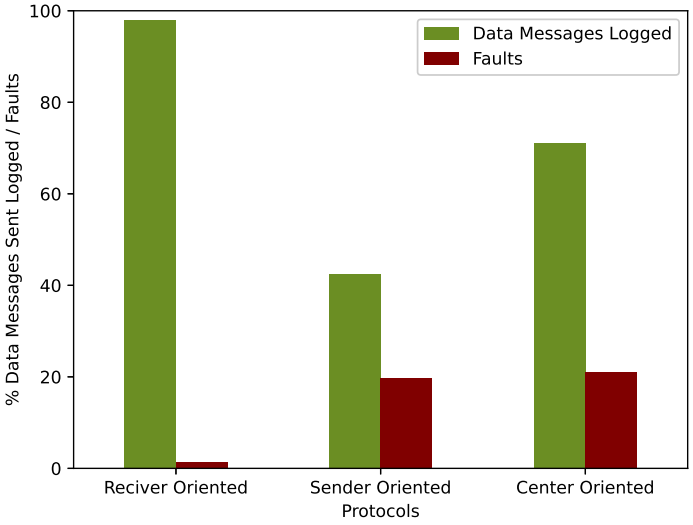


Figure 5.18: A Comparison Between the Performance of the Sender, Receiver and Centre Oriented on the FIT IoT-Labs.

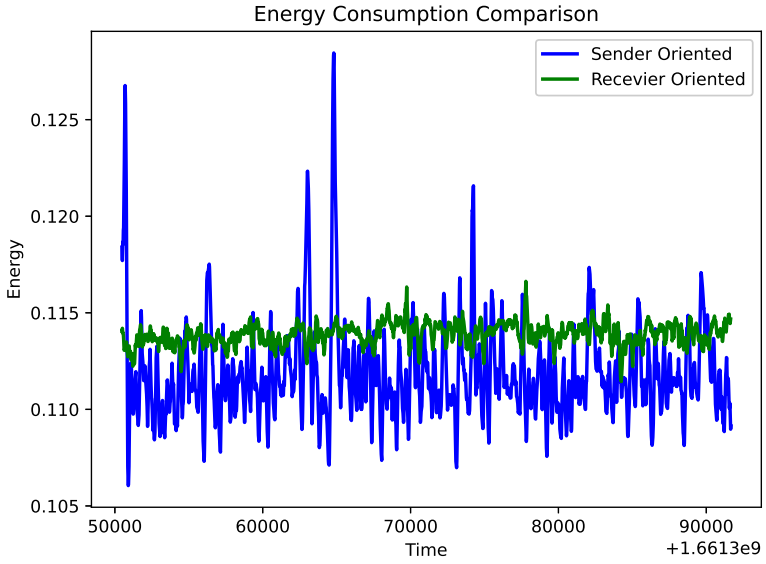


Figure 5.19: A Comparison Between the Average Energy Consumption in the Sender and Receiver Oriented on the FIT IoT-Labs.

deployment.

- Considering the overall picture, the *sender oriented* seems to perform more efficiently than the*receiver oriented*; however, there is a significant number of energy spikes in the *sender oriented*.

As nodes in the *sender oriented* transmit their control messages to the data message sender logger, and as the chance of that logger being more than one hop distant from the sender node's neighbour is **high**, more energy is required to route these control messages.

While *receiver oriented* nodes transmit control messages to their perspective logger and the chance that the logger is more than one hop distant is **low**, we see constant energy usage throughout the span of deployments.

We have a complete picture when we include the *centre oriented* average energy consumption over time. From Figure 5.20 we observe the following:

- Compared with the other version of the *Collector* algorithm, the *centre oriented* energy consumption is lower.
- While we can observe several low points in the overall energy consumption of the *centre oriented*, the energy spikes are the average consumption.

For context, we also include the typical energy requirements for a run of the

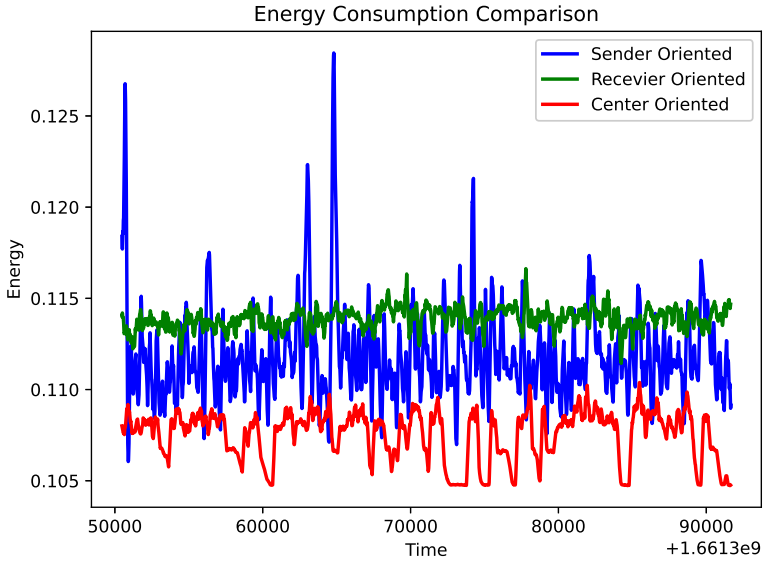


Figure 5.20: A Comparison Between the Average Energy Consumption of the nodes in the Sender, Receiver and Centre Oriented on the FIT IoT-Labs.

RPL protocol under the same settings as those used in the other tests, Figure 5.21. The following inferences can be drawn from the figure:

- Compared to the typical run of the network, all the *Collector* algorithm versions seem to consume higher energy.
- While the *centre oriented* seems to do better than the other two versions, it still consumes higher energy than RPL.

The *Collector* algorithm in all versions, *sender oriented* and *receiver oriented*, as well as the *centre oriented*, requires more energy than a standard run of RPL; this is intuitive. However, adding additional processes would eventually imperil some performance metrics due to the inherent characteristics of networks for the Internet of Things.

In our situation, although it appears that the *Collector* method consumes more energy, the cost is negligible compared to the benefits we would obtain.

5.7 Discussions

Numerous issues influence the design of any protocol for an IoT wireless sensor-based network; these range from network states to node capabilities and include a variety of parameters which contribute to the volatile nature of IoTWS

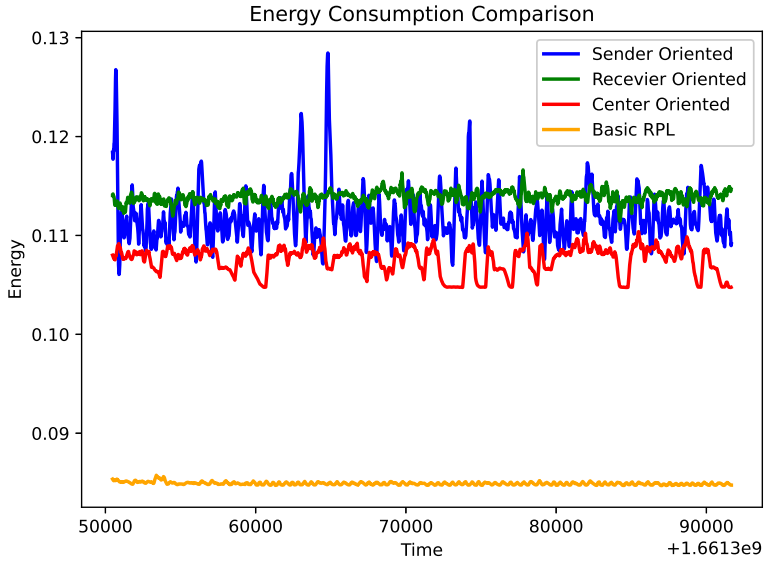


Figure 5.21: A Comparison Between the Average Energy Consumption of the nodes in the Sender, Receiver, Centre Oriented and RPL Basic on the FIT IoT-Labs.

networks. Keeping track of network events is no exception. We demonstrated in this study that collecting logs about nodes' activity in IoTWS networks required an upper bound on the number of Byzantine nodes and $2k + 1$ network connectivity, with each node having at least $3f + 1$ neighbours. The preceding section discussed the behaviour of these parameters in simulations and experiments. This section discusses a few other factors affecting the collector's performance.

5.7.1 Faults Number

The amount of faults in the network is the starting point; obviously, the more faulty nodes in the system, the more faults will arise. The collector algorithm is not an exception in this respect. The number of faults continues to grow, and in some circumstances, the number of faults is the same as the number of logged messages. As illustrated in Figure 5.22, as the number of faulty nodes increases, the number of logged faults also increases, whereas the amount of logged messages are unaffected; therefore, we continue to have correctly logged messages.

5.7.2 Faults Types

A system may face a variety of different types of defects; the one that we replicated in our experiments is an omission fault. An omission fault occurs when a node violates the protocol by failing to send control messages following the

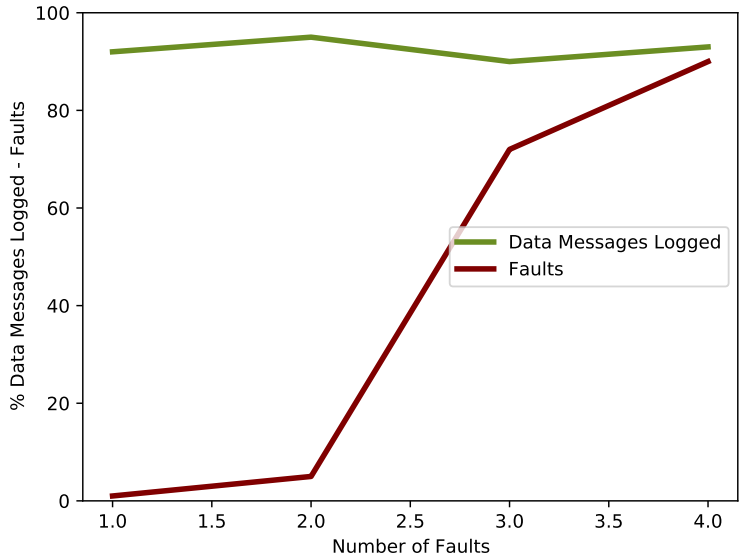


Figure 5.22: The Number of Faults Increasing with the Number of Faulty Nodes in the Network.

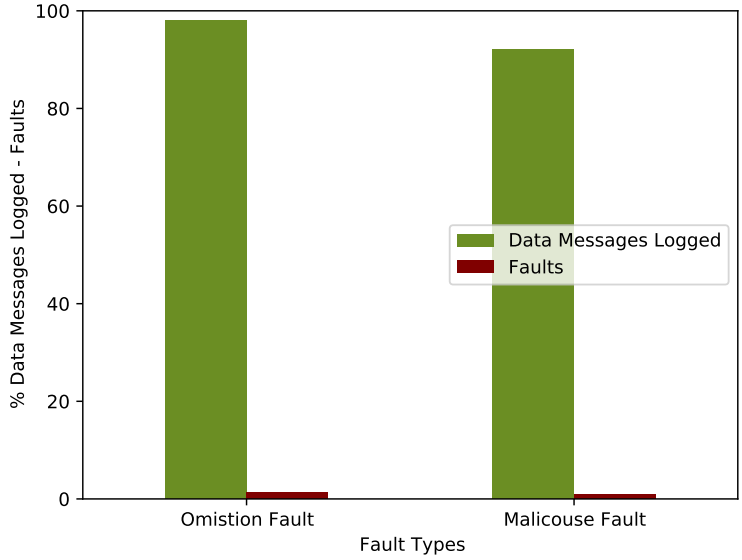


Figure 5.23: The Receiver Oriented Performance with Different Types of Faults.

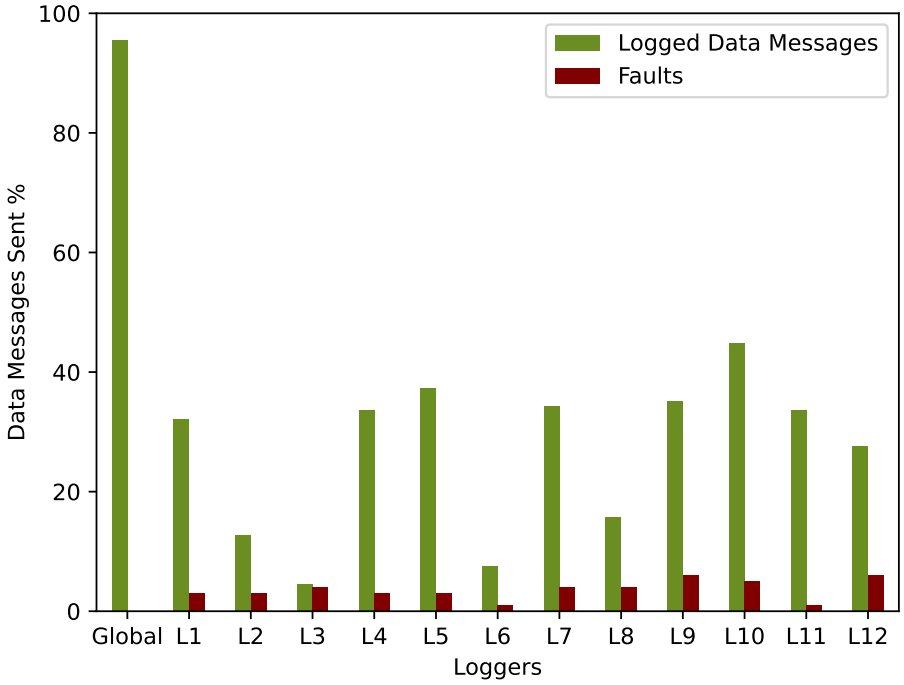


Figure 5.24: Global Analysis vs Local Analysis Receiver Oriented

transmission of a data message to the sink.

We simulated a malicious fault to obtain a more accurate evaluation of the collector's performance. A malicious fault occurs when a node transmits erroneous control messages. This leads to the creation of a fault log.

We examined the *receiver oriented* performance in both faults in Figure 5.23. With identical nodes, topology, and one faulty node, it is evident that regardless of the type of fault, the *receiver oriented* logs practically all data messages sent with negligible failures in both circumstances.

Additionally, the fault type used in Figure 5.22 is malicious faults. Though we see an increase in the number of faults as faulty nodes increase, we still have practically all data messages transmitted logged.

5.7.3 Logging Hierarchy

Typically, data are collected and stored in permanent storage that can hold them for longer. In certain instances, data are collected on local machines, but at some point in their life cycle, they are pushed to a central unit that stores the acquired data for the entire system.

Consequently, the result given in this paper is an analysis of globally collected data. As is customary, we collected data collected by loggers and stored them in large log files before analysing them. Analysing locally collected data will yield somewhat different results due to other factors, such as the neighbourhood size and the number of faulty nodes within that neighbourhood; however, this should be fine as logs will eventually be pushed into more permanent storage.

As one would expect, the number of faults per logger increases when analysing logs locally. The case is evident when we look, in particular, at the *receiver oriented* analysis in Figure 5.24. While the number of logged messages varies per logger due to the different cluster sizes, we can also notice various faults per logger.

As nodes send their control messages to their prospective logger, the logger will not receive the required number of messages and, therefore, record a fault. However, we can see that when we globally analyse the logs, we do not record any faults as all the messages are in one place rather than in several places.

5.8 Conclusion

This chapter examined how to conduct reliable logging in the Internet of Things wireless sensor-based networks in the presence of Byzantine faults. We defined and demonstrated that the logger selection problem is NP-Complete.

Then, we described the collection problem, showed impossibility results, and provided the strong, weak and probabilistic variants. After the collection algorithm's presentation, simulation and testbed deployments were conducted.

We found that while the sender-oriented algorithm should, in principle, provide the most significant number of correctly logged messages, in practice, it performs poorly due to collisions. In addition to having the lowest energy consumption in the testbed deployment compared to the other versions, the receiver-oriented approach yields the best results regarding the logged messages and faults.

Chapter 6

MAC Platform Effects on Reliable Logging

Previously, we discussed the event logging (collection problem), outlined the difficulties of log collecting in a network with Byzantine nodes, and showed several impossible results for such a scenario. Weak (log) collection was one of the problems we presented, and we demonstrated how it might be overcome in some settings, such as when the network is divided into clusters and cluster members only inform the cluster head (i.e., the logger) of events occurring inside their cluster. With a CSMA-based MAC protocol, we verified the efficacy of our suggested approach. As one of the most used MAC protocols, we opted for the carrier sense multiple access (CSMA) MAC layer as it provides a simple method of managing data transmissions.

However, unreliable communication was problematic since it prevented the loggers from accurately recording network events. Loggers failed to capture events because messages were being delayed by network contention, leading to an abundance of false positives (wrong failures) being recorded. Here, we look at the weak log collecting problem from the perspective of reliable communication, specifically as it relates to the TSCH-based MAC layer that was used to solve it and which is discussed in more detail in the following chapter.

In the Industrial Internet of Things (IIoT), where the uptime of mission-critical applications relies on such networks, reliable communication is of the utmost importance. Time is segmented into frames in TSCH networks, and inside each frame are many time slots, each of which can be used to transmit data via a different channel. Afterwards, each node broadcasts at its designated time slot on its designated frequency. TSCH networks may be considered a sort of TDMA/FDMA hybrid. Consequently, the network contention is less.

While CSMA-based networks provide unique challenges, TSCH networks do

as well. Thanks to a shared timetable, the nodes can conserve energy by knowing when to turn on and off their radios in advance. However, there are drawbacks to being able to save energy in this way; for instance, a node may delay the delivery of a message because the radio is switched off, which might lead to the message being overwritten if a new message is received before the old one has been processed in a full buffer (due to resource constraints).

Therefore, to recap, the issue we are trying to solve in this section: Is it possible to implement weak collection (See Definition 4)(i.e. reliable event logging) on a TSCH network where nodes have limited available resources?

6.1 Addition to Models

This chapter builds on the preceding models used in the previous results, to which we add the models below that are essential to the findings shown in this chapter:

6.1.1 Slot Assignment

As this chapter focuses on time slotted channel hopping (TSCH) based networks, we assume that at least one time slot has been given to each node for data transmission.

Definition 9 (Converging Slot Assignment)**.** *Given a network $G(V, E)$, a slot assignment $\mathcal{A} : V \to \mathbb{N}$, a path $n_0 \cdot n_1 \ldots n_{i-1}$ between a source node $n = n_0$ and a sink $\Delta = n_{i-1}$ is called* slot converging for n to Δ *if* $\forall\ j, 0 \leq j < i - 1, \mathcal{A}(n_j) < \mathcal{A}(n_{j+1})$. *$\mathcal{A}$ is called* convergent to Δ *if $\mathcal{A}$ is slot converging for all nodes $n \in V$.*

When a slot assignment $\mathcal{A}$ is convergent, it guarantees that a message will reach the sink within one time period. Please note that a slot assignment implies that only a single slot is assigned to each node.

We do not assume that nodes know about the global slot assignment, i.e., the network schedule. A node is on (i.e., not sleeping) when it is sending a message. It *may* also be up when one of its neighbours is sending (potentially to receive the message). We then explain duty-cycling as the process during which a node goes from the on-state to the off-state (sleep) and vice-versa.

6.1.2 Fault Model

This section continues to follow the previously explained notion of Byzantine behaviour; however, in the light of the TSCH MAC, we will include additional Byzantine actions. Since nodes turn off their radios on a schedule, a Byzantine node

could act as a correct sleeping node and so either sleep when a message is received or it may send a message in the incorrect slot or fail to receive or send in their active slots.

6.2 TSCH Schedulers

As previously described in 3.1.3.2, the *Time Slotted Channel Hopping (TSCH)* MAC protocol is a combination of *Time Division Multiple Access (TDMA)* and *Frequency Division Multiple Access (FDMA)* in the sense that TSCH combines the time division property of TDMA protocols with the frequency division property of FDMA.

Therefore, the TSCH scheduler is constructed based on the channel hopping value, which specifies the range and number of channels that nodes are permitted to communicate on and the number of time slots. Creating a matrix where each slot is described by its own channel offset and time offset.

TSCH typically utilises two sorts of slots: broadcast slots and unicast slots. Typically, the broadcasting slot is reserved for EB and protocol-related control messages, i.e. RPL control messages, whereas the unicast slot is for data transfer. These slots might expand based on the implemented scheduler and can be customised to be devoted to a specific message type in accordance with the schedule's criteria.

This TSCH flexibility enabled the development of many schedulers that favour network optimisation on one side over the other. We have 6TiSCH [202] that allows IPv6 on top of TSCH; Orchestra [77] that allows nodes to build their own schedulers based on the RPL information; ALICE [119] that builds the scheduler based on the link quality between nodes; OST [106] that optimises TSCH to provide better reliability in a network with mobile nodes.

Literature has an extensive number of TSCH schedulers, each of which focuses on increasing TSCH performance in a particular context. However, the most used schedulers are 6TiSCH and Orchestra, which we employ to explore the challenge of reliable logging in a TSCH environment. In this part, we comprehensively describe how these schedulers function.

6.2.1 6TiSCH

TSCH uses different schedulers to manage the scheduling process, with the default scheduler being 6TiSCH. 6TiSCH enables the use of IPv6 in addition to TSCH. It allows for the benefit of IPv6 in addition to TSCH and supports node and schedule management through a CoAP interface. While IEEE 802.15.4 defines how the scheduler operates, it does not specify how the scheduler is built. Thus,

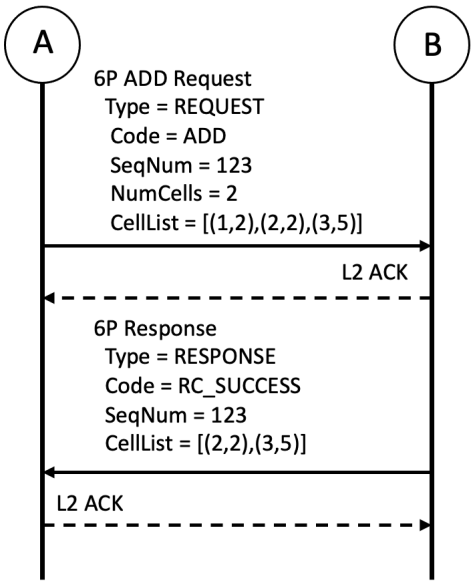


Figure 6.1: A 2-step 6top Transaction Example [202]

6TiSCH specification provides minimal configurations and builds the scheduler in a distributed manner [202].

6TiSCH is managed by two components: (i) a Scheduling Function and (ii) the 6TiSCH operational sub-layer *6top*. 6top enables nodes to negotiate the addition of slots between them based on the available links in the 6TiSCH scheduler; the node adds a new link if there are insufficient slots and deletes the extra links if there are no messages to send.

6top:

6top includes two types of transactions, (i) a 2-step (see Figure 6.1) and (ii) a 3-step. The 2-step transaction comprises a REQUEST phase and a RESPONSE phase, while the 3-step transaction also includes a CONFIRMATION phase, Figure 6.2. A REQUEST message can either be an ADD (a request to add a new slot allocation), DELETE (a request to delete an allocation) and, CLEAR (to reset the negotiation differently), a RESPONSE message could either be SUCCESS or ERROR. If it is a 3-step transaction, the CONFIRMATION phase start by setting a timeout after receiving a SUCCESS until a link-layer acknowledgement is received to confirm the transaction; if the timeout expires before receiving the acknowledgement, the transaction will be dropped.

A transaction starts with node A initiating a REQUEST message of type ADD to request a new slot allocation; node B accepts the request by replying SUCCESS. If a message is lost in the transaction, node A will retransmit after a

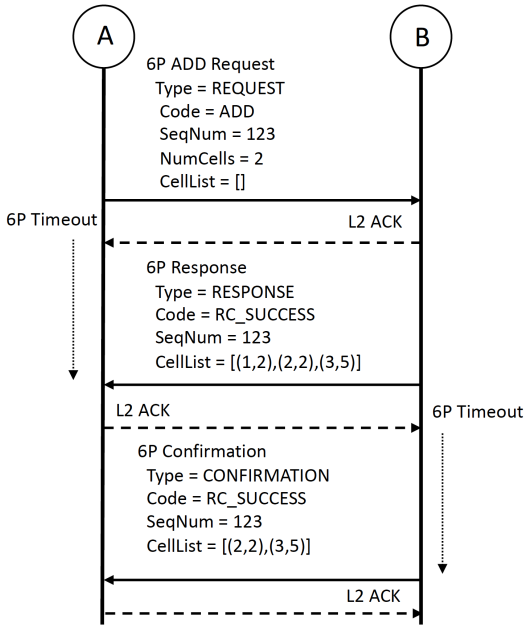


Figure 6.2: A 3-step 6top Transaction Example [202]

predefined timeout.

6top messages include a sequence number for record-keeping that allows node B to detect a missing or lost message and issue an ERROR message to which node A replies with CLEAR to reset the schedule. The policy that governs the 6top negotiations and slot assignments is defined in the Scheduling Function. A scheduling function can be designed in such a way as to optimise network performance or manage different types of communications. 6TiSCH has standardised the ***Minimal Scheduling Function (MSF)*** [199].

MSFs provide two types of cells, namely (i) autonomous cells and (ii) managed cells. The managed cells respond to the network traffic, and nodes use the autonomous cells to define the minimal bandwidth with their neighbours. A quick note on command transactions: **6p** commands are transmitted in Information Element packets, which travel a single hop only [198]. Once the node joins the 6TiSCH network, the node will add, delete or reallocate cells based on one of these three reasons [56]:

1. To match the link-layer resources.
2. To handle schedule collisions.
3. To handle parent switches.

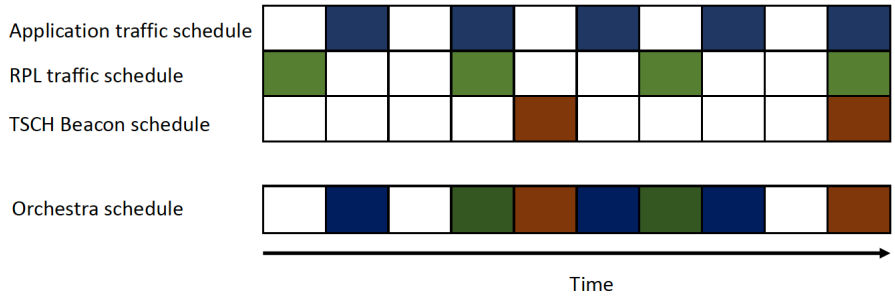


Figure 6.3: Orchestra Scheduling[77].

6.2.2 Orchestra

Orchestra is another scheduler that makes use of TSCH and RPL. It differs from other schedulers because it does not require negotiations, a central entity, signalling or multi-hop path reservation. Each node maintains its schedule locally based on its RPL parents and neighbours.

Orchestra comprises various size slot frames, each of which is dedicated to a specific type of traffic, such as MAC (TSCH beacons), routing (RPL), and application data (see Figure 6.3).

Nodes choose slots based on scheduling rules designed to reduce contention, making Orchestra ideal for event-based applications [77]. Orchestra defines four main slots:

- ***Common Shared Orchestra Slots (CS)***: This is a slot that is shared by all nodes in the network for the Tx (transmission) and Rx (reception) and is installed with fixed coordinates.
- ***Receiver-Based Shared Orchestra Slots (RBS)***: These slots are assigned to two neighbours based on the properties of the receiver coordinates, e.g., a child-to-parent communication. Since several nodes install Tx slots for the same receiver, contention may occur in these slots.
- ***Sender-Based Shared Orchestra Slots (SBS)***: SBS are similar to RBS, except that slot coordinates are assigned using the sender nodes properties instead of the receiver. SBS slot results in one Rx slot per neighbour coordinates based on the neighbour, and a single Tx slot, coordinates based on the sender.
- ***Sender-Based Dedicated Orchestra Slots (SBD)***: SBD is similar to SBS but uses dedicated TSCH slots, not shared.

Orchestra uses different slots for each node, and the length of the slot is connected with various network metrics. The shorter the slot gets, the more traffic

the network will have; it affects the per-hop network latency and energy consumption. Shorter slots mean that the period will repeat, often consuming more energy.

Orchestra maintain its scheduler based on a set of *scheduling rules*. The rules are a combination of TSCH slot frames and slots with enhanced Orchestra properties. The properties of the slots in Orchestra are: Handle, Length, Traffic Filter, Neighbours, Coordinates and Options.

6.3 Theoretical Results

In this section, we present our theoretical results.

In the context of TSCH-based networks, we introduce a novel weak collection variant called the *fast weak collection*, which we define below.

Definition 10. *(n-Fast Weak Collection Problem) A programme is a solution for the fast weak collection problem if each of the programme's computations satisfies the following properties:*

- ***Safety**: At a correct logger l, DM_l is a subset of the set of data messages that have been sent, i.e., $DM_l \subseteq \mathcal{D}$.*
- ***Timed Termination**: A correct logger l will either log a control message $\langle CM(m,s)\rangle$ for a data message $\langle(m,s)\rangle$ sent or a fault message $\langle FM(m,s)\rangle$ within n periods.*
- ***Validity**: A fault message $\langle FM(m,s)\rangle$ at a logger l is logged only if there are faulty nodes in the network.*

The first result concerns the feasibility of the weak collection in a (duty-cycled) TSCH-based network[1].

Theorem 6.3.1 (Impossibility of Weak Collection)**.** *Given a network $G(V,E)$, with $|V| = \gamma$, a set $\mathcal{L} = \{L_1, \ldots, L_n\}$ of loggers, each logger L_i related to a set of nodes $n_i^1, \ldots, n_i^l$, at most k of Byzantine nodes in any neighbourhood where $V \leq 2k+1$ and a duty cycled schedule $\mathcal{D} : V \times T \to \{0,1\}$, then there is no algorithm that can solve the weak collection problem.*

Proof. We prove this by contradiction: We assume the existence of an algorithm $\mathcal{A}$ that solves the weak collection problem and then show a contradiction.

Assume a run c_1 in which (i) a node n sends a data message M in round r, (ii) $\forall m \in V, \forall t \in T \cdot D(m,t) = 1$ and (iii) $\exists V' \subseteq V, |V'| = k$, where nodes in V' are Byzantine nodes and they do not send or forward control messages to some logger

[1]Whenever we say a TSCH network, we mean a duty-cycled TSCH network.

L_j. Assume that L_j records a fault for M under A in c_1, i.e., $\mathcal{A}$ records $F(M, n)$ in L_j.

Now, assume a run c_2 which is similar to c_1 as follows: a node n sends a data message M in round r. However, c_2 differs from c_1 as follows: (i) no node behaves maliciously in c_2 and (ii) $\forall n \in V', \forall t \in T \cdot D(n, t) = 0$, i.e., all nodes that were Byzantine in c_1 are now asleep all the time. Logger L_j receives the same set of messages in c_2 as in c_1. Since $\mathcal{A}$ is assumed to be correct, it logs a fault $F(M, n)$ in L_j. However, no Byzantine nodes in c_2 violated the validity property.

Hence, $\mathcal{A}$ does not exist. □

Weak collection cannot be satisfied due to the fact that sleeping nodes and Byzantine nodes may induce similar behaviours in a duty-cycled IoT environment.

The next issue to be addressed is whether fast weak collection is possible when nodes are allocated a single slot.

Theorem 6.3.2 (Impossibility of 1-Fast Weak Collection)**.** *Given a network $G(V, E)$, with $|V| = \gamma$, a set $\mathcal{L} = \{L_1, \ldots, L_n\}$ of loggers, each logger L_i related to a set of nodes $n_i^1, \ldots, n_i^l$, a convergent slot assignment $\mathcal{A} : V \to \mathbb{N}$, then there is no algorithm that can solve the 1-fast weak collection problem.*

Proof. We assume the existence of such an algorithm. We show that such an algorithm cannot exist through the construction of an appropriate structure. To prove this, we need to show that $\mathcal{A}$ is generally convergent for all loggers.

We focus on the data message sink Δ and a logger L_j for the control message. Consider a node n and a path $n \cdot n_1 \ldots \Delta$ with $n = L_j$. Since $\mathcal{A}$ is convergent, the path for node n_1 to Δ is also slot converging.

Now, assume node n_1 to be in the cluster of logger L_j. Since the path $n \cdot n_1 \ldots \Delta$ is slot converging, the path $n_1 \cdot L_j$ is not *slot* converging; this means the control message will travel from node n_1 to node n in the following period. Hence, $\mathcal{A}$ will be at least 2-fast weak collection.

This is a contradiction.

□

What is required is that a path to the sink and also paths to loggers are all slot converging. For this to happen, nodes need to have more than a single slot, i.e., nodes need to have multiple slots, or loggers need to be carefully chosen so that paths to both loggers and sink are slot converging; this is captured in the following result.

Corollary 6.3.2.1 (1-fast weak collection)**.** *Given a network $G(V, E)$ with sink Δ and with $|V| = \gamma$, a set $\mathcal{L} = \{L_1, \ldots, L_n\}$ of loggers, each logger L_i related to a*

set of nodes $n_i^1, \ldots, n_i^l$*, at most* k *of Byzantine nodes in any neighbourhood, a slot assignment* $\mathcal{A} : V \to \mathbb{N}$*, it is possible to solve the* 1*-fast weak collection problem if* $\mathcal{A}$ *is convergent to sink* $s \in \mathcal{L} \cup \{\Delta\}$*.*

TSCH-based networks automatically assign single slots to nodes. This means that 1-fast weak collection is impossible to guarantee, especially when loggers can be located randomly in the network. As such, to ensure that slot assignment is convergent, what is needed is an approach whereby nodes can negotiate for more slots.

We thus propose an algorithm that extends any single slot assignment algorithm to negotiate for other slots for control message routing to loggers. Since we assume nodes know the schedule, the negotiation algorithm (See Algorithm 4) proceeds as follows:

1. Denote the node that needs to send the control message to the logger by s and the logger by L. Denote the data message slot for s by d_s, the node schedule S.

2. A path between s and L is known by s due to routing protocol. Denote the path by $s \cdot n_1 \ldots L$.

3. Node s requests to *add* a new slot, denoted by c_s to n_1. c_s needs to be greater than d_s, as a control message will be sent after receiving a data message. Node s proposes c_s to node n_1.

4. Since node n_1 is not up in slot c_s, then n_1 sends a *success* message.

5. This process is repeated until the logger is reached and all nodes on the path between s and L agree to add new slots.

6. When achieved, a schedule convergent to L will be obtained.

If the slot c_s that node s is requesting from neighbour n_1 is already scheduled for another transmission in n_1 schedule, node s will keep the control message in its buffer while attempting to renegotiate a slot with either n_1 or any other neighbour that is on the same path to its logger l_s. However, as node s attempts to negotiate the control message slot, it will continue to receive or prepare other messages for transmission according to its schedule.

Before these messages are handled, they are kept in the buffer of node s in a packet queue. If this buffer is full and node s needs to add a new packet to the queue, the oldest packet in the queue will be overwritten so that the new packet may be processed. If the node buffer is full before the node can secure a slot, the node s

Algorithm 4 ***Slotted Collector***

s: control message sender;
S: Node schedule.
L: list of loggers, l_s is node s logger;
d_s: node s data message slot;
$s.n_1..l_s$: the path between node s to l_s;
c_s: control message slot for node s;

repeat
req (c_s, n_1);
  if $c_s \notin S_{n_1}$ **then** *success*
  else *fail*
  end if

until $c_s \in S_{l_s}$

will discard the control message in this situation. We call to this a buffer overflow [158]. As IoTWS nodes have limited storage capacity by design, buffer overflow is a regular occurrence, which in this situation has a more significant impact on TSCH MAC than the CSMA MAC setting; hence, we offer this conclusion.

Theorem 6.3.3 (Impossibility of 1-Fast Weak Collection with finite $buff\_size$)**.** *Given a network $G(V, E)$, with $|V| = \gamma$, a set $\mathcal{L} = \{L_1, \ldots, L_n\}$ of loggers, each logger L_i related to a set of nodes $n_i^1, \ldots, n_i^l$, at most k of Byzantine nodes in any neighbourhood, a convergent slot assignment $\mathcal{A} : V \to \mathbb{N}$, a duty cycled schedule $\mathcal{D} : V \times T \to \{0, 1\}$, then there is no algorithm that can solve the 1-fast weak collection problem if $\forall n \in \{n_i^1, ..., n_i^l\}$ related to L_i has a finite $buff\_size(n)$.*

Proof. We prove this by contradiction: We assume the existence of an algorithm $\mathcal{P}$ that solves the 1-Fast Weak Collection with finite buffer size and then show a contradiction.

Assume a computation c_1 in which (i) a node n sends a data message M in round r, (ii) $\forall n \in V, buff\_size(n) = finite$ and (iii) $\exists V' \subseteq V, |V'| = k$, where nodes in V' are Byzantine nodes and they do not send or forward control messages to some logger L_j. In this case, L_j will receive less k control messages and will records a fault for M under P in c_1, i.e., $\mathcal{P}$ records $FM(M, n)$ in L_j.

Now, assume a computation c_2 which is similar to c_1, as follows: a node n sends a data message M in round r. However, c_2 differs from c_1 as follows: (i) no node behaves maliciously in c_2, all Byzantine nodes in c_1 act correctly this round and send a control message to L_j. However, as n neighbours receive the control message and start negotiating a slot with L_j, they receive other messages, and they have to drop the control message before completing the negotiations. Logger L_j receives the same set of messages in c_2 as in c_1. Since $\mathcal{P}$ is deterministic, it will log a fault $FM(M, n)$ in L_j. However, there were no Byzantine nodes in c_2, which is

contradictory.

Hence, $\mathcal{P}$ does not exist. □

6.4 Evaluation

In this section, we explain our experimental setup for both simulations and deployment on the FIT-IoT lab. We discuss the schedulers used for slot assignment and the clustering techniques.

6.4.1 Schedulers and Clustering

As clustering is half of the collection problem's setup and RPL is our primary routing protocol, we wanted to determine whether the clustering algorithm impacted the logging problem during these tests. This section explains how the clustering method influences the performance of the *CollectorCollector* algorithm.

6.4.1.1 RPL and Clustering

Before examining the sender-oriented algorithm's performance over other TSCH schedulers, it is essential to evaluate the impact of clustering in RPL over the TSCH MAC environment. As indicated previously 5.2, clustering is used to choose loggers in the network. However, in 5.6, we did not employ a specific clustering technique; instead, we focused on the log collection problem and, as our tests were performed in a CSMA environment, focusing on the clustering algorithm seemed superfluous.

In contrast, this is not the case in TSCH settings. Orchestra relies on the RPL neighbour table and parents to design the schedule, which is hampered by the fact that RPL routing does not support clustering that is not directed to the sink [79]. More specifically, RPL only supports clustering where control messages flow in the same direction as the data message that eventually will reach the sink.

A prominent cause of packet loss in downward routing (i.e., higher level node to lower level node in the DODAG) is MAC layer drop, which occurs when the MAC layer drops a message after the maximum number of retransmission attempts has been reached. While this does not affect the CSMA configuration, when Orchestra is applied, packets are lost because the nodes in the path do not hear a message (i.e., deaf) or are Byzantine, which, as previously indicated, is impossible to determine.

The second reason is that of the inconsistent routing; when a packet is sent to a destination that is not the parent of the sender or the sink, it will travel up until a common ancestor (of sender and destination) is reached and then gets routed to the destination; if there is no shared ancestor, it travels to the sink and then gets routed

to the destination [211], this is where the problem of link asymmetry manifests itself; control messages travelling to a logger that is more than one hop away and not a potential parent is lost throughout the transaction.

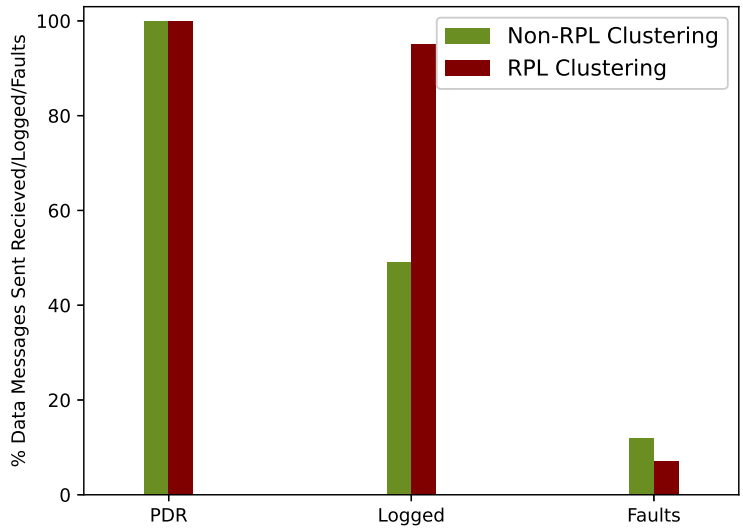


Figure 6.4: The Sender Oriented Performance in Different Clustering Algorithms In CSMA.

6.4.1.2 RPL and non-RPL Clustering

We use two types of clustering in this section. In non-RPL-based clustering, the clusters and cluster heads, i.e., the loggers, are predetermined independently of the routing. In RPL-based clustering, on the other hand, the cluster heads are determined by the nodes with the most significant number of children.

By leveraging RPL structures, the path from nodes to loggers will be slot converging, unlike when loggers are randomly chosen. The impact of such clustering on message routing for log collection is detailed below.

In the log collection implementation used in this chapter, nodes send control messages to the data message sender's logger (cluster head). While the probability of all control messages being sent to the same logger is high, the probability that this logger is multiple hops away from the sender of the control messages is also high, hence increasing the probability of message loss (see section 5.6.2.2).

To better understand this effect, we provide a result for the receiver-oriented variation of the *CollectorCollector* algorithm. In this variant, the nodes send their control messages to their respective logger; hence, the probability that the logger is

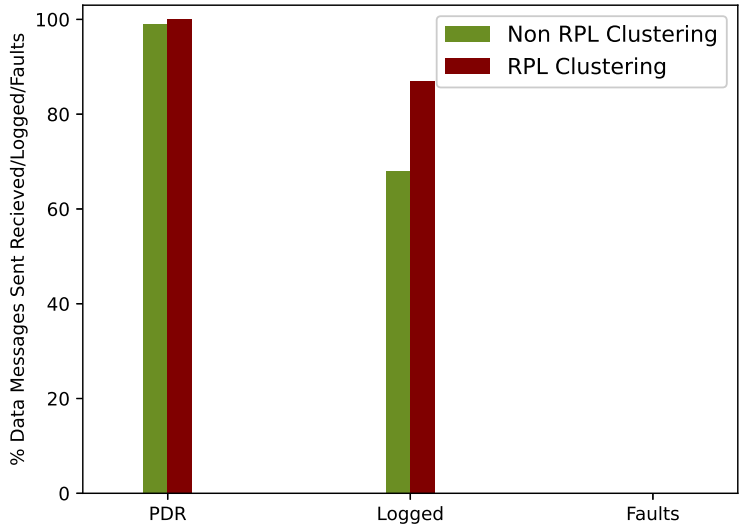


Figure 6.5: The Sender Oriented Performance in Orchestra Different Clustering Algorithms.

one hop away is high.

Specifically, we evaluate the performance of Orchestra with multiple configurations and clustering algorithms using the receiver-oriented version of the collector algorithm.

From the Figure 6.6, we can observe the following:

- Regardless of the Orchestra rule or clustering algorithm, the delivery rate of data messages is consistently high.
- Regardless of the Orchestra rule, non-RPL-based clustering performs poorly, considering the number of logged messages and faults.
- The sender-based rule yields superior results compared to the recipient-based rule.

6.4.2 Simulations Setup

To validate our results and algorithms, we performed several simulation experiments, running the sender-oriented collection algorithm over several MAC protocols, namely (i) CSMA, (ii) TSCH with 6TiSCH and 6top and (iii) TSCH with Orchestra. We ran the simulations using Cooja, a simulator with the Contiki-ng operating system [2], and RPL [192] as the primary routing protocol used in our simulations according to our system model.

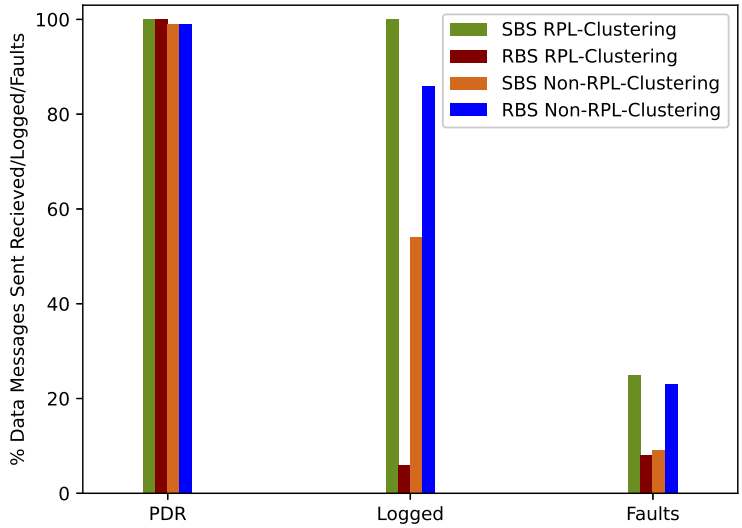


Figure 6.6: The Receiver-Oriented Performance in Different Orchestra, Clustering Algorithms.

We ran several simulations for each scenario: (i) a sink as a root for the network, (ii) a set of loggers as cluster heads, (iii) 10% of the network as sending nodes and (iv) $(n-1)/3$ Byzantine nodes, where n is the number of nodes.

6.4.3 Simulations Results

6.4.3.1 CSMA

We ran the sender-oriented algorithm in this scenario with the default CSMA settings. Nodes send their control messages to the sender-based logger. Figure 6.4 shows that sender-oriented performs perfectly with an almost packet delivery ratio (PDR) for data messages and all the logged messages. However, in Figure 6.10, we see that sender-oriented consumes more energy in CSMA mode than in any other scenario, as nodes are always awake.

6.4.3.2 Orchestra

Orchestra produces very interesting results. Although it provides a near-perfect delivery ratio, the amount of messages logged is much lower than is the case in the sender-oriented CSMA protocol. The Orchestra protocol employs a set of rules that the nodes must adhere to when creating their schedule. The default

implementation of Orchestra in Contiki-ng is as follows:

```
#define ORCHESTRA_RULES{&eb_per_time_source,\
                        &unicast_per_neighbor_rpl_storing,\
                        &default_common }
```

The first part of the rule is related to the settings of the TSCH and RPL EB messages; the second is dedicated to unicast messages between the children and their parents; lastly, the common default slots are for broadcasts and other unicast that are not scheduled in the second part of the rule.

To obtain the optimal outcome for the sender-oriented collector version, we modified the following Orchestra configuration parameters:

- in place of the RPL non-storing mode, which is the default configuration for Orchestra implementation in Contiki-ng, we opted for the RPL storing mode because loggers can be located at multiple hops distances.
- Since control messages are not scheduled in advance and must be scheduled based on the node, the sender-based rule seemed to be the most natural choice.
- We also increased the number of common shared slots and unicast periods.
- The hash function was also configured to avoid collisions.

The outcomes are depicted in Figure 6.5, to which we can attribute the following:

- Regardless of the clustering algorithm employed, the data message delivery ratio is almost optimum.
- Compared to the RPL-based clustering, the non-RPL clustering results in a much lower volume of logged messages due to Orchestra generating slot converging paths by leveraging RPL.

As Orchestra allocates slots dynamically, when a node has control messages to send according to the sender rule, Orchestra will assign a slot based on the node's coordinates for the control message.

6.4.3.3 6top

Here, we discuss the results of implementing the 6TiSCH 6top layer. In this implementation, we utilised the scheduling function, a simple function supplied by Contiki-ng. We employed this function as a *black box* to establish its impact on the logging process.

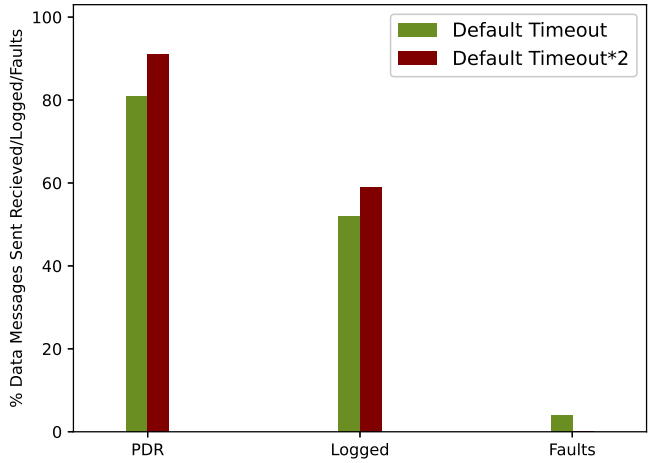


Figure 6.7: The Sender Oriented 6top Results with Different Timeouts

Not only does it perform poorly regarding logged messages, but the data messages delivery ratio falls significantly compared to other MAC implementations, with PDR ratios of less than 90% and correctly logged messages as low as 60%, making it the version with the least efficient performance.

As with Orchestra, we increased the default timeout to determine if modifying a setting might improve 6top's performance; this was done specifically due to the fact that when a node sends a request to schedule a slot, the scheduling function sets a timeout to complete the negotiations; if the timeout expires prior to the conclusion of the negotiations, the transaction is cancelled.

Therefore, we decided to double the simple function's existing timeout. As seen in Figure 6.7, this change did improve the results slightly, but less than the other MAC versions; this is quite counter-intuitive as nodes now may have two slots and are given a higher time to complete the negotiations. However, we observed that, by increasing the timeout, the node buffers became full, and message overwrites started to happen.

Using 6top for the *Collector*'s implementation may have seemed like a good approach in retrospect, but this is not the practice case. A more intelligent scheduling function must be developed to enable the negotiation to schedule data and control message slots for each node.

6.4.4 Experiments Setup

In this section, we look into the performance of the *sender oriented* in CSMA, Orchestra and 6top in the Fit IoT LAB. In similar software settings as the simulations,

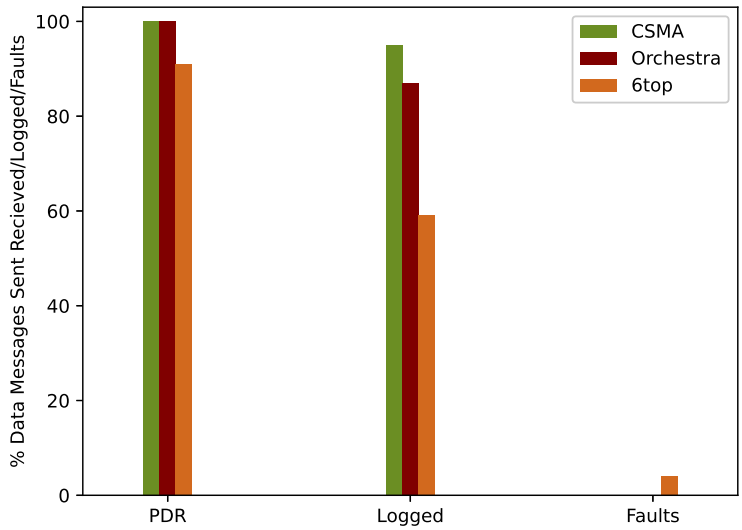


Figure 6.8: The Performance of Sender-Oriented Algorithm in Different MAC settings using Cooja Simulator.

but on the M3 nodes in the Lille site in Fit IoT LAB, we ran several deployments to test the algorithm's performance and measure the nodes' average energy consumption.

6.4.5 Experiments Results

The results of the deployment on FIT IoT-LAB are not significantly different from those of the Cooja simulator. From Figures 6.8 and 6.9, the following may be deduced:

- While CSMA provides ideal performance in Cooja owing to its virtual environment, collisions reduce the amount of logged messages dramatically in FIT IoT-LAB since all control messages are routed to the same logger simultaneously.
- Due to the presence of Byzantine nodes, we observe a rise in logged faults despite the fact that Orchestra performance in FIT IoT-LAB needs to catch up in terms of PDR and logged messages.
- 6top is similar in FIT IoT-LAB, but despite providing a large amount of logged messages, its PDR falls well short of Cooja simulations, though it still compares well with Orchestra.

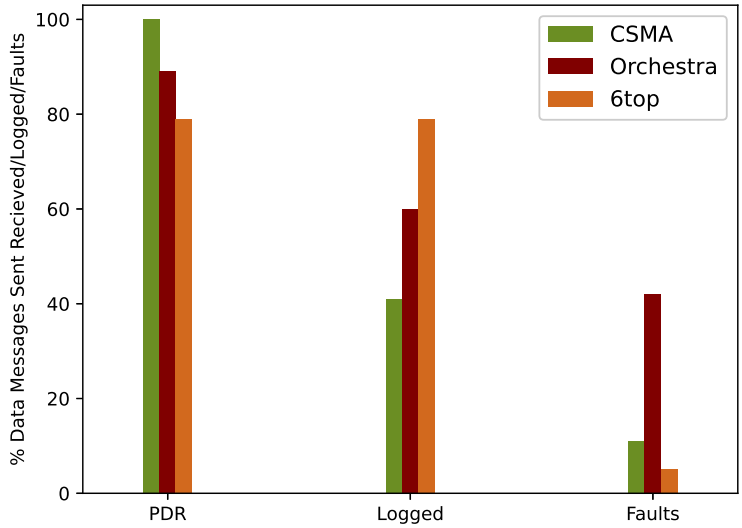


Figure 6.9: The Performance of the Sender-Oriented Algorithm in Different MAC Settings in FIT IoT-Lab.

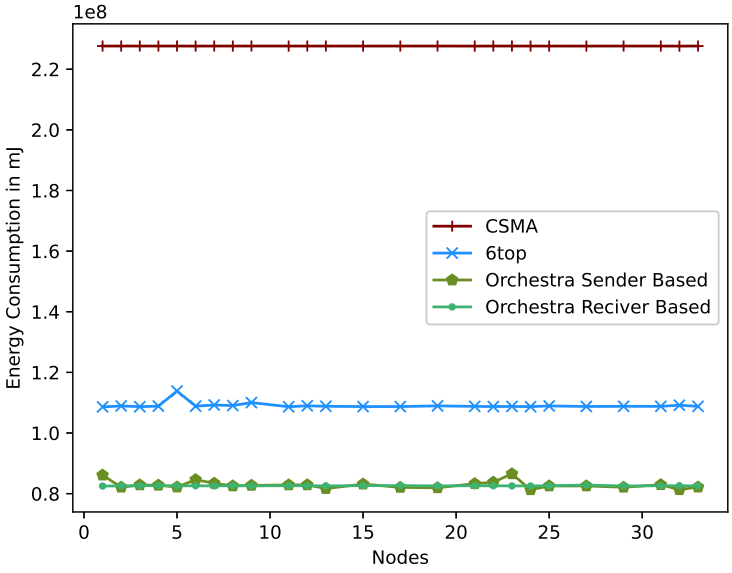


Figure 6.10: Per Node Total Energy Consumption of the Sender Oriented in CSMA, 6top and Orchestra in Cooja Simulator.

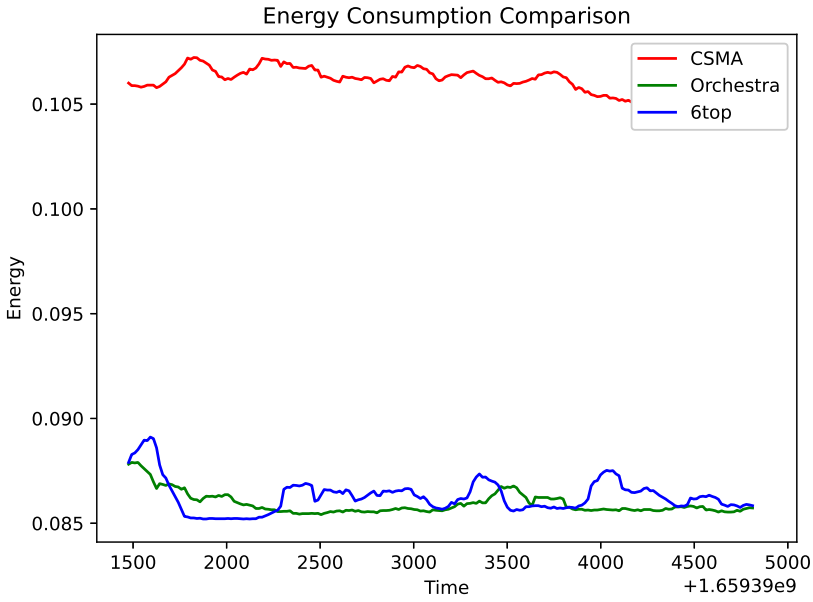


Figure 6.11: Over Time Average Energy Consumption of the Sender Oriented in CSMA, 6top and Orchestra in Fit IoT-Lab.

6.4.6 Energy Consumption

One of the primary benefits of utilising a TSCH scheduler is the potential energy efficiency gain for IoT wireless sensor-based networks. Consequently, one of the most important metrics that must be examined is the energy usage under each scheduler compared to the "on-all-the-time" CSMA.

In order to accomplish this, we utilised Contiki-ng's Energest module to obtain energy metrics. Figure 6.10 reveals that, as predicted, CSMA has the highest energy consumption, whereas Orchestra, with both sender-oriented and receiver-based rules, has the lowest energy consumption. 6top requires substantially less energy than CSMA but still more than Orchestra (due to negotiations).

The deployments in the FIT IoT-LAB yield comparable outcomes. CSMA continues to consume the most, while 6top and Orchestra consume the same rate on average (see Figure 6.11).

6.5 Conclusion

While the present TSCH schedulers provide dependable service and performance in various network scenarios, their performance has yet to be assessed in the presence of Byzantine network faults and their impacts.

In this chapter, we developed the 1-fast weak collection variant to solve the

problem of reliable logging within a TSCH MAC setting. We demonstrated that it could only be solved by having slot converging paths to the sink and loggers, each node requiring multiple slots in a period.

We demonstrated the performance of Orchestra and 6TiSCH with the *Collector* algorithm, and we saw that unless new slots were added to the schedulers, they fall behind compared to the CSMA setting.

Chapter 7

Towards Auditable System Through Reliable Logging

As we mentioned in earlier chapters, in IoTWS networks, logging occurs at the sink; however, due to the network size, the logs have to travel through multiple hops causing message loss and network contention, which makes satisfying the completeness property of the logs challenging to achieve.

We then proposed logging the events distributedly by dividing the networks into clusters, with each cluster reporting to its cluster head, a.k.a logger. We used histories to ensure the nodes were reasonably far from the loggers to prevent message loss. However, we have noticed that with Byzantine faults, it is not always the case that we can have a correct log for every event in the network.

In this chapter, we look into another case of logging where every node is a logger. This scenario includes nodes recording their interactions with other nodes and any relevant data. As all the nodes act as loggers, we only focus on the logs between a pair of nodes.

The nodes will keep specific logs, which means that depending on the application, the information that nodes will log will differ. In the case of a service provider, for example, the nodes will log the request, the resource sought, and the resource specification. Depending on the type of network, this resource will vary; it could be related to the link quality or a specific routing metric. Nevertheless, ensuring that each node logs the complete interaction is challenging, particularly if we consider the presence of malicious behaviours.

In this chapter, our problem statement can be summarised as follows: In a network with Byzantine nodes, where every node is a logger, there is an algorithm that ensures that the generated logs are either correct or if it is not correct, a fault can be detected.

Towards this goal, we look into the equivalent notion of auditability. As in auditablity, an auditor examining a log file will verify the log's accuracy or identify falsehoods and inconsistencies.

Formally, we can define the auditing process as the process of acquiring and subsequently analysing data on system execution; a system that enables this analysis through logged data is known as an *auditable* system.[1]

Auditing is a versatile approach that can be applied to various circumstances, including (i) fault detection and (ii) digital forensics, among others. Audit trails, which are chronological sets of system data indicating the order in which events (or states) occurred during execution, are typically generated from the data collected during an auditing process, also known as histories or correct logs. Audit trails include cluster system logs and debugging data, for example. These documents provide evidence of specification compliance. [2] An auditable system generally records relevant information on its operation and is capable of thorough analysis.

We study the notion of auditability, and we investigate the question of whether we can enforce the completeness property through auditability.

Towards this objective, we make the following contributions in this chapter:

- We define the auditability problem and the concept of an auditable programme.
- We demonstrate that a system can only be made auditable for a class of properties referred to as safety properties.
- We concentrate on constructing auditable distributed programmes and demonstrate several impossibility results.
- We demonstrate that, for a given system model, the problem of strong auditability is analogous to the problem of strong fair exchange.
- We conclude with a case study in an IoT wireless sensor-based network context demonstrating that auditability is, in fact, fundamental to the algorithmic design of sensor networks.

This chapter is organised as follows: in 7.1, we add the additions that are needed for the system model for this particular problem; in 7.2, we define the auditability problem; in 7.3 we present our impossibility results, and in 7.4 we set the auditability requirements and provide an example. Next, in 7.5, we show that the auditability problem is as difficult as the fair exchange problem. In 7.6, we introduce the case study of the RPL protocol as an auditable programme and conclude in 7.7.

[1] Auditability is the property of a program that enables it to be auditable

[2] We will hereafter define auditability as the ability of a programme to gather and log execution details of ongoing processes reliably.

7.1 Additions to Models

Besides the models presented in 3.2 we add the following models to provide the basis for this chapter's contributions:

7.1.1 Properties and Safety Properties

We define a property $\rho \subseteq \mathcal{S}^*$ as a set of states sequences, i.e., a trace σ satisfies a property if $\sigma \in \rho$. Hence, a property is defined exclusively in terms of individual computations.

Properties that specify that "nothing bad ever happens" are called *safety properties*. A property Γ is a safety property for a system with states set $\mathcal{S}$ if $\forall \sigma \notin \Gamma, \exists \sigma' \sqsubseteq \sigma, \forall \alpha \in \mathcal{S}^* : \sigma' \cdot \alpha \notin \Gamma$

Informally, this definition states that once a bad action has taken place, thereby excluding the initial part of an execution from the property, there is no extension of that prefix that can remedy the situation, i.e., once something bad has happened, it cannot be undone.

Intuitively, using the example of mutual exclusion 7.4.2, it means that, for example, if two processes simultaneously acquire the locks for the critical section, then even if later, one of the processes relinquish the lock, the fact remains that safety was violated.

7.1.2 Specification

A specification is a set of computations. Programme ϕ *satisfies* specification $\mathcal{SPEC}$ if every computation of ϕ is in $\mathcal{SPEC}$.

Alpern and Schneider [5] stated that every computation-based specification can be described as the conjunction of a safety and liveness property. Intuitively, a safety specification states that something bad should not happen, whereas a liveness specification states that something good will eventually happen.

Formally, the safety specification identifies a set of finite computation prefixes that should not appear in any computation. A liveness specification identifies a set of computation suffixes such that every computation has a suffix in this set.

A specification $\mathcal{SPEC}$ is a set of computations. A programme $\mathcal{P}$ is a set of computations that satisfies a $\mathcal{SPEC}$ iff $\mathcal{P} \subseteq \mathcal{SPEC}$. A predicate ω is a subset of $\mathcal{S}$: $\omega = \{s | \omega(s) = True\}$.

7.1.3 Fusion Closure

The set $\mathcal{S}^*$ is said to be *fusion closed* if for all traces $\sigma_1 = \alpha.s.\kappa \in \mathcal{S}^*$ and $\sigma_2 = \eta.s.\chi \in \mathcal{S}^*$ then $\alpha.s.\chi \in \mathcal{S}^*$ and $\eta.s.\kappa \in \mathcal{S}^*$.

Here, $\alpha, \kappa, \eta, \chi$ are sequences. We call s a fusion point of the two traces [25]. Fusion closure basically states that the history information of an execution exists at every step of the system.

7.2 Problem Statement

An auditable system will have audit trails that will allow an auditor to examine the functionality or performance of the system via an analysis of the system's controls. The logs can be used to *detect* any suspicious behaviour or as a *prevention* tool for bad behaviours in the future[98]. In an open distributed system, enforcing auditability is challenging as an attacker may not be willing to log the requests being made.

In this chapter, we consider a system or programme $\mathcal{P}$ that comprises two parties or processes: (i) a service provider P and (ii) a service requester R. R sends resource requests to P, which then allocates the requested resources to R if the request is good. If R is an honest party, then R requests and uses the resources according to its specification. In this work, only malicious faults are considered; crash faults are not accounted for. Similarly, if P is honest and R's request is good, then P will service the request.

On the other hand, if R (or P) is a malicious participant, the sequence of requests from R (or service provisions from P) may be malicious. Thus, an auditable system will keep audit trails of the interactions between P and R to detect misbehaviour.

Thus, if S is an unauditable system, it must be transformed into an auditable one. Thus, we define the auditability problem as follows:

Definition 11 (Auditability Problem)**.** *Given a program $\mathcal{P}$ with two processes R and P, a resource ψ, a specification $\mathcal{SPEC}$ for ψ, find a transformation T : such that $T(\mathcal{P})$[3] satisfies the* transparency *and* auditability *properties, i.e., $\forall \sigma \in traces(\mathcal{P})$:*

1. *Transparency: if σ satisfies $\mathcal{SPEC}$, then $T(\sigma) = \sigma$.*
2. *Detection: if σ violates $\mathcal{SPEC}$ then: T: $\exists \langle \sigma' \rangle \sqsubseteq \langle \sigma \rangle, \langle \sigma' \rangle$ violates $\mathcal{SPEC}, T(\sigma) = \sigma'$.*

The transparency property states that if (honest) party R generates a good sequence $\langle r \rangle$, then R will not change its behaviour even if the system is auditable. Then, P also provides resources according to the requests by R, i.e., $\langle p \rangle \models \langle r \rangle$.

[3]We extend the notation of T here to mean all elements of the set P.

On the other hand, if (malicious) party R generates a bad sequence in the absence of T (i.e. when the system is not auditable), then when the system is auditable, R can still generate bad sequence $\langle r \rangle$, but the system can detect when something bad occurred, i.e., when $\langle r' \rangle$ occurred, but P does not have to guarantee.

Given a program $\mathcal{P}$, a resource ψ with specification $\mathcal{SPEC}$, we say that $\mathcal{P}$ is $\mathcal{SPEC}$-*auditable* for ψ (for short, we say *auditable* for ψ when $\mathcal{SPEC}$ is clear from the context) if, for all traces $\sigma \in \text{traces}(\mathcal{P})$, either:

- σ satisfies $\mathcal{SPEC}$.
- σ violates $\mathcal{SPEC}$, then $\exists \sigma', \sigma' \sqsubset \sigma, \sigma'$ violates $\mathcal{SPEC}$.

A programme is auditable for resource ψ if either a trace is good or if the trace is not good, it returns the violating prefix of the trace.

7.3 Impossibility Results

What is then required to satisfy the auditability property is the programme's ability to retain execution history. The auditability property is easily satisfied in a system where processes are trusted (or honest). Processes exchange information about occurring events so that the history information is recorded.

However, satisfying the auditability property when the processes are malicious (or dishonest) is challenging, as those malicious processes may keep the wrong history or may even tamper with the history information. In a system where malicious processes are *not* previously known, it is impossible to keep the historical interactions between a requester and server processes.

7.3.1 Impossibility of Auditability of Distributed Programs

We consider the auditability problem in the following setting: A programme $\mathcal{P}$ consists of two processes, one process R, called a resource requester and P, a resource provider. P is asynchronous, meaning that there is a finite but unknown delay on both system execution and message transfers. At least one of P and R is dishonest. When R makes a request, it may record its request r in its history. On the other hand, when P receives r, it may record r and, when it provides the resource to R, it may record the service s being provided. When R receives a notification about s, it may record s. We assume both P and R know $\mathcal{SPEC}$.

We consider auditability as a service, similar to broadcast primitives developed in [90]. We consider two primitives:

- $log(m)$: A process p makes a call to $log(m)$ to signify its intention that the interaction involving m be recorded.
- $record(m, x)$: A call to $record(m, x)$ enables a process p to be notified of the fact that information for a given message m is complete and the item (m, x), where x is the corresponding interaction for m has been recorded in its history.

A request is made via a message, just as a service is provided via a message, in the sense that a process is allowed to access a resource via a token, similar to the situation in a token ring network. Please observe that x can be $\perp$, meaning there was an undefined message. For example, a message with $\perp$ means the message was wrong or the service was not provided. Otherwise, a message can come from a well-defined set of messages. We call a message where either $m = \perp$ or $x = \perp$ an *abort* item or a fault log, as in the case of the log collection problem.

We now specify the formal properties of the *strong auditability*[4] problem:

- **Termination:** Every correct process *eventually* executes $record(m, x)$.
- **Validity:** If a correct process p executes $record(m, x)$, either (m, x) satisfies $\mathcal{SPEC}$ or the item is an abort item.
- **(Strong) Fairness:** At the end of the protocol, either both processes execute $record(m, x)$ or both of them record an abort item.

The termination property captures that correct processes must record that they have processed an item, good or bad. The validity property captures the protocol's behaviour when the processes behave correctly: a correct process will only record an item when the item is good; else will record an abort if the interaction violates $\mathcal{SPEC}$.

On the other hand, the fairness property captures the intended behaviour when at least one process behaves malevolently: that either both processes record a correct item or both of them record an abort item.

We now present our second result.

Theorem 7.3.1 (Impossibility in Asynchronous System)**.** *There exists no deterministic algorithm that solves the auditability problem when $\mathcal{P}$ is asynchronous, and processes are malicious.*

Proof. We assume that such an algorithm exists and then show a contradiction.

Assume an execution E of the system, where all processes behaved well, with the following characteristics:

[4]Henceforth, whenever we say auditability problem, we mean strong auditability.

- process P executes $log(m)$ at time t_1.
- process R executes $log(r)$ at time t_2.
- process P executes $record(m, r)$ at time $t_3 > t_1$.
- process R executes $record(r, m)$ at time $t_4 > t_2$.

Now, consider an execution E', similar to E, in which R is malicious, executes $record(r, m)$ but does not execute $log(r)$. So, at t_3, P can do the following: (i) Execute $record(m, \perp)$ (i.e., an abort item) to satisfy termination property or (ii) wait for r to arrive to and eventually execute $record(m, r)$, to satisfy fairness. In the first case, to satisfy termination, fairness is violated. In the second case, to satisfy fairness, termination is violated, as r will never arrive at P.

By circular reasoning (i.e., where P is malicious, instead of R), we conclude that both processes need to execute $log(m)$ and $log(r)$ at the same time. This means that the system is synchronous, which is a contradiction.

□

7.3.2 Auditability and Trusted Process/TTP

Since solving auditability is impossible in an asynchronous network, we focus on designing auditable systems in a synchronous system. Unfortunately, doing this is challenging again, as shown by our following result.

Theorem 7.3.2 (Impossibility without TTP)**.** *There exists no deterministic algorithm that solves the auditability problem when $\mathcal{P}$ is synchronous with malicious processes and without a trusted process in the system.*

Proof. We assume that such an algorithm exists and then show a contradiction.

Assume an execution E of the system, where all processes behaved well, with the following characteristics:

- process P executes $log(m)$ at time t_1.
- process R executes $log(r)$ at time t_1 - as the system is synchronous (following Theorem 7.3.1).
- process P executes $record(m, r)$ at time $t_3 > t_1$.
- process R executes $record(r, m)$ at time $t_4 > t_1$.

Now, consider an execution E', similar to E, in which R is malicious, executes $record(r, m)$ but executes $log(r')$ and (m, r') violates $\mathcal{SPEC}$. So, at t_3, P will execute

$record(m, \perp)$ (i.e., an abort item) to satisfy termination property but violating fairness. Since r is only held by R, and that r cannot be created; it means that fairness and termination cannot be satisfied simultaneously, leading to the impossibility result. □

To satisfy both the fairness and termination properties A process needs to be able to recreate r to satisfy both the fairness and termination properties simultaneously. Since the network is assumed to satisfy the integrity property, it cannot create r, meaning there is a process with a copy of r that will not tamper with it, i.e., a trusted process.

7.4 Auditability Requirements and Examples

7.4.1 Auditability Requirements

We seek to answer the following question: For which type of specification is a programme $\mathcal{P}$ auditable?

Theorem 7.4.1 (Auditability Requirements)**.** *Given a programme $\mathcal{P}$, a resource ψ with specification $\mathcal{SPEC}$, $\mathcal{P}$ is $\mathcal{SPEC}$-auditable for ψ only if*

1. *$\mathcal{SPEC}$ is a safety specification.*
2. *$\mathcal{SPEC}$ is fusion closed.*

Proof. 1. $\mathcal{P}$ is $\mathcal{SPEC}$-auditable.

$\Rightarrow \forall t \in \text{traces}(\mathcal{P}), t \in \mathcal{SPEC} \vee (t \notin \mathcal{SPEC} \Rightarrow \exists t' \sqsubseteq t : t' \notin \mathcal{SPEC})$.

$\Rightarrow \forall t \in \text{traces}(\mathcal{P}), t \in \mathcal{SPEC} \vee (t \in \mathcal{SPEC} \vee \exists t' \sqsubseteq t : t' \notin \mathcal{SPEC})$

$\Rightarrow \forall t \in \text{traces}(\mathcal{P}), t \in \mathcal{SPEC} \vee \exists t' \sqsubseteq t : t' \notin \mathcal{SPEC}$

$\Rightarrow \forall t \in \text{traces}(\mathcal{P}), t \in \mathcal{SPEC} \vee \exists t' \sqsubseteq t, \forall t'' : t' \cdot t'' \notin \mathcal{SPEC}$

$\Rightarrow \mathcal{SPEC}$ is a safety specification. □

2. $\mathcal{P}$ is $\mathcal{SPEC}$-auditable.

$\Rightarrow \forall t \in \text{traces}(\mathcal{P}), t \in \mathcal{SPEC} \vee (t \notin \mathcal{SPEC} \Rightarrow \exists t' \sqsubseteq t : t' \notin \mathcal{SPEC})$.

We assume $\mathcal{SPEC}$ not to be fusion closed and show a contradiction. Let two traces be: $t_1 = \alpha.s.\kappa \in \mathcal{SPEC}, t_2 = \chi.s.\eta \in \mathcal{SPEC}$.

$\mathcal{SPEC}$ is not fusion closed $\Rightarrow \alpha.s.\eta \in \mathcal{SPEC} \wedge \chi.s.\kappa \notin \mathcal{SPEC}$.

$\mathcal{P}$ is $\mathcal{SPEC}$-auditable $\Rightarrow \forall t \notin \mathcal{SPEC}, \exists t' \sqsubseteq t : t' \notin \mathcal{SPEC}$. Let $t' = \chi.s$

Since $\chi.s \notin \mathcal{SPEC}$, and $\mathcal{SPEC}$ is a safety property, $\forall \varrho$, ϱ being a sequence, $\chi.s.\varrho \notin \mathcal{SPEC}$.

But, $t_2 \in \mathcal{SPEC}$, leading to a contradiction. Thus, $\mathcal{SPEC}$ is fusion closed.

□

The first result indicates that the specification needs to be a safety specification. Specifically, the specification for the resource to be shared needs to be a safety specification for a programme to be auditable for that resource. This result overlaps with that of [170], which says that the class of security policies that is EM-enforceable is safety property. In a sense, auditability is a special case of the EM class as it requires the programme to either display good behaviours or else a violating prefix can be identified.

However, our second result means that history information needs to be available in every state of the programme for it to be auditable. In a sense, this is very intuitive: if history information is not kept about a resource usage, then it becomes very challenging to determine any specification violation [61]. In other words, using the log collection problem we can transform an un-auditable system into an auditable system.

7.4.2 Auditability Examples

Having discussed the specification requirements, we now consider an example to illustrate the auditability problem. Consider two possible specifications for mutual exclusion:

- A process executes in the critical section only when the critical section is free (Algorithm 5).
- A process executes in the critical section only when the critical section is free and when some fairness condition is satisfied (Algorithm 6).

To better illustrate the auditability problem, we consider the mutual exclusion problem for a system consisting of n process $P_1, P_2, P_3 \ldots P_n$; Each process has a code region, called a critical section, in which they access (read or write) shared data structures. To ensure consistency of the data structure, access to the data is mediated by using a mutual exclusion algorithm. There are various mutual exclusion algorithms. While all of them provide exclusive access (ME property) to the data structure, some of them will provide additional properties such as fairness (see Algorithm 5) [181].

Algorithm 5 ***Mutual Exclusion (ME) Algorithm with No Fairness (NF) Property (ME-NF)***

```
lock: global variable ∈ {TRUE, FALSE}.
do{
    while (TestAndSet(&lock))
        ; \\ Do Nothing
        \\ Critical Section
        lock = FALSE;
        \\ Reminder Section
}while(TRUE);
```

As observed in [24], every non fusion closed specification can be converted into an equivalent one with the addition of a history variable. Hence, ME is fusion closed. As can be observed, algorithm ME-NF is ME-auditable as all of its executions will be correctly classified.

On the other hand, consider a different specification for mutual exclusion, namely bounded waiting mutual exclusion (BW-ME). Again, BW-ME is fusion closed. In this case, algorithm ME-NF is BW-ME-auditable as, this time, some of its execution will be classified as bad and the violating prefix can be identified when an interaction that violates BW-ME is found. On the other hand, algorithm ME-F (Algorithm 6) is also BW-ME-auditable and, in this case, all the executions are good.

To prevent a race condition situation, only one process can access its critical section at a time. A lock is needed to facilitate this solution. Algorithm 6 shows the code of the process with the critical section. Before entering the critical section, the process needs to set the lock to true, indicating that it is currently executing the critical section to the other process. Once the process finishes executing the critical section, it set the lock to false indicating that the other processes are free to enter the critical section.

In other words, as long as the processes behave accordingly, each process will have its time executing the critical section, thus achieving the transparency property of the auditability problem.

However, this solution could also result in endless waiting for other processes. If P_1 for example, is faster than P_2, it could enter the critical section far more than P_2, which results in unfairness execution for P_2. To solve this problem, a time limit should exist to limit the amount of time the processes wait between requesting to enter the critical section and accessing the critical section.

The algorithm in 6 implements the concept of bounded waiting. The process enters the critical section if either the key or the waiting variable is false. The key becomes false once the process executes $TestAndSet()$, while the waiting becomes

Algorithm 6 ***Mutual Exclusion (ME) Algorithm with Fairness (F) Property (ME-F)***

$lock$: global variable $\in \{TRUE, FALSE\}$.
key: the critical section waiting variable.
$waiting$: the process waiting variable.
$do\{$
    $waiting[i] = TRUE;$
    $key = TRUE;$
    $while$ ($waiting[i]$ && key)
        $key = TestAndSet(\&lock)$
    $waiting[i] = FALSE;$
    $\backslash\backslash\ Critical\ Section$
    $j = (i + 1)\%n;$
    $while$ $((j \neq i)$ && $!waiting[j])$
        $j = (j + 1)\%n;$
    If ($j == i$)
        $lock = FALSE$
    Else
        $waiting[j] = FALSE;$
    $\backslash\backslash\ Reminder\ Section$
$\}while(TRUE);$

false only for one process at a time once the process exits the critical section. After it is done, it searches the array for the next process which has the waiting variable = true as the next one to enter the critical section, thus ensuring that all processes enter the critical section within $n - 1$ rounds.

Put differently, the processes need to keep a log of all the requests to enter the critical section, preventing other processes from misbehaving and detecting what processes did misbehave, thus satisfying both of the auditability properties.

7.5 Auditability and Fair Exchange

We studied the auditability problem as a programme in which processes can be divided into resource requester and resource provider. In order to transform this programme into an auditable one, the resource request must be logged, together with the request's status and the nature of the available resources. In other words, the entire transaction must be recorded.

From a service provider's perspective, this resembles the idea of a fair exchange. Informally, a fair exchange involves two parties exchanging items fairly, in which participants supply information about the item they are requesting and a description of the exchanged items while documenting the entire transaction.

Consequently, we examine strong auditability through the lens of strong fair exchange. We begin with a quick overview of fair exchange before presenting our theoretical findings.

7.5.1 Fair Exchange

This section formally defines the fair exchange problem and its specification based on the previous literature. As mentioned earlier, there are several conceptualisations of the fair exchange problem; in this work, we use the notion of ***strong and weak fairness***.

According to [26], a protocol solves the fair exchange problem if it satisfies the following properties:*Effectiveness, Fairness, Timeliness* and *Non-repudiation*. Given two participants m and n with items I_m (with description D_m) and I_n (with description D_n) respectively, then a protocol P satisfies these properties:

- *Effectiveness:* If m and n behave correctly and do not want to abandon the exchange, then when P has completed, m will have I_n such that I_n will satisfy D_n (resp. I_m and I_m satisfies D_m)
- *Fairness:* Two main notions of fairness exist:
  - ***Strong Fairness:*** When the protocol P terminates, either m (resp. n) has I_n and I_n satisfies D_n (resp. I_m and I_m satisfies D_m) or none of the participants has any information about each other's item.
  - ***Weak Fairness:*** when the protocol P terminates, either m (resp. n) has I_n and I_n satisfies D_n (resp. I_m and I_m satisfies D_m) or none of the participants has gained information about each other ***Or*** m (resp. n) can prove to a judge that n (resp. m) has received, or still can receive, I_n (resp. I_m) such that I_n satisfies D_n (resp. I_m satisfies D_m).
- *Timeliness:* m Can be sure that P will be completed at a certain point in time. At completion, the state of the exchange will be either final or changed into a state that does not degrade the level of fairness reached so far
- *Non-repudiation:* When the exchange is terminated, P can prove:
  - ***Non-repudiation of Origin:*** the origin of I_m (that it was originated from m).
  - ***Non-repudiation of Receipt:*** that m received I_n.

7.5.2 Strong Auditability and Strong Fair Exchange

In certain respects, strong auditability is analogous to strong fair exchange. As explained previously, the fair exchange problem involves two participants exchanging two items where both receive their desired items; if the exchange is completed successfully, otherwise the exchange will be aborted. Similarly, auditability involves two participants exchanging a request and a service provision (and then recording them), or neither records anything. The setting in which strong auditability can be solved is a synchronous system with a trusted process.

Lemma 7.5.1 (Solving Strong Fair Exchange using Strong Auditability)**.** *Given a synchronous system model with two processes and a trusted process and also given a programme $\mathcal{A}$ that solves strong auditability in this system model. Then, there is a programme $\mathcal{F}$ that solves strong fair exchange in the same system model.*

To show this, we provide an algorithm that solves strong fair exchange using an algorithm for strong auditability, which will then show that strong auditability is at least as difficult as a strong fair exchange.

Algorithm 7 ***Strong Fair Exchange Algorithm using Strong Auditability - Process*** $i \in \{R, P\}$

$item_i$: process i exchanged item.
$\mathcal{SPEC}$: $item_i$ specification.
e_R: Expected returned item. $\{*Contains\ item_j*\}$
status: local variable, $\in \{abort, 1\}$.
Function: strong_fair_exchange (item$_i$, $\mathcal{SPEC}$)
    status := *strong_auditability* ($item_i$, $\mathcal{SPEC}$)
    If ($e_R = record(item_i, item_j) \wedge item_j \neq \perp$))
        return ($item_j$) ▷ SA returns (e_R)
    Else
        return (*abort*) ▷ SA returns ($\perp$)

Proof. We prove two cases: (i) when processes execute properly and (ii) when processes are malicious.

- Case (i): When both processes are good, then, when strong-auditability terminates, both processes execute $record(itemA, itemB)$ for processes A, B. Process A then retrieves itemB for the fair exchange protocol. A similar circular reasoning applies to process B. Hence, fair exchange's termination, fairness and validity properties are also satisfied.

- Case (ii): The only time strong auditabilty can return "abort" is when the item received does not match the description (i.e., wrong item or item not received).

When strong auditability aborts, both processes execute $record(abort)$ when they terminate (satisfying strong fairness), thus, the processes return" abort" for fair exchange.

□

Lemma 7.5.2 (Solving Strong Auditability using Strong Fair Exchange)**.** *Given a synchronous system model with two processes and a trusted process and given also a programme $\mathcal{F}$ that solves strong fair exchange in this system model. Then, there is a programme $\mathcal{A}$ that solves strong auditability in the same system model.*

We now provide an algorithm that solves strong auditability using an algorithm for strong fair exchange.

Algorithm 8 ***Strong Auditability Algorithm using Strong Fair Exchange - Process*** $i \in \{R, P\}$

m: The message sent from R or P.
$\mathcal{SPEC}$: *auditability* specification.
e_R: Expected returned item.
status: local variable, $\in \{abort, 1\}$.
Function: $strong\_auditability\ (item_i, \mathcal{SPEC})$
    $status := strong\_fair\_exchange\ (item_i, \mathcal{SPEC})$
    If ($status \neq abort$)
        $return\ (\ record(m, e_R)\ )$ ▷ SFE returns $item_j$ *in* e_R
    Else
        $return\ (record(m, \perp))$ ▷ SFE returns (***abort)***

Proof. We prove two cases: (i) when processes execute properly and (ii) when processes are malicious.

- Case (i): When both processes are good, then when strong fair exchange terminates, both processes have their respective desired items, i.e., process A having itemB and process B having itemB. At this point, process A has both items and records them in its history. A similar circular reasoning applies to process B. Hence, the termination, fairness and validity properties of strong auditability is satisfied.

- Case (ii): The only time strong fair exchange can return "abort" is when one or more of the items received do not match the description (i.e., wrong item or item not received). When strong fair exchange aborts, both processes return *abort* when they terminate (satisfying strong fairness), thus, the processes return "record(abort)" for strong auditability.

□

We assume that calls to *strong-fair-exchange* (Algorithm 7) and *strong-auditability* (Algorithm 8) are executed *atomically.*

Finally, we present this result.

Theorem 7.5.3 (Equivalence of Strong Fair Exchange and Strong Auditability)**.** *Given a synchronous system model with two processes and a trusted process. There exists a programme $\mathcal{F}$ that solves strong fair exchange in this system model iff that there is a programme $\mathcal{A}$ that solves strong auditability in the same system model.*

Proof. This follows simply from Lemmas 7.5.1 and 7.5.2 □

7.6 Case Study

The previous sections show that strong auditability and strong fair exchange are interchangeable, i.e., that strong auditability can be attained by a strong fair exchange. In addition, we demonstrated that at least one process must be trusted in a malicious environment to solve strong auditability.

We demonstrate this using the RPL routing protocol as an example, to show how auditability underlines the design principles of wireless sensor network algorithms. RPL, as previously stated, is an IETF-standard distance-vector protocol for IP-based communications in low-power wireless. Despite the fact that RPL is designed for networks with lossy links, we construct the experiments so that the links are trustworthy. Please keep in mind that, while wireless communication can be asynchronous at times, WSNs are designed to perform several retransmissions when messages are lost. The timeout in a synchronous system assumption is taken to be the time until all retransmissions have occurred.

RPL is a protocol that is based on the formation of a destination-oriented directed acyclic graph (DODAG), where the paths are constructed towards the DODAG root. The RPL node discovery mechanism uses three types of messages:

1. DODAG information object (DIO), which is used to discover the DODAG.
2. Destination Advertisement Object (DAO) is sent by the nodes to disseminate destination information in the DODAG.
3. DAG information solicitation (DIS) is sent to discover the DODAG and encourage a DIO from the nearby nodes

RPL employs an objective function F to calculate the routing routes in RPL to generate the DODAG. To determine the optimum routes in the network, the

```
#if !RPL_MRHOF_SQUARED_ETX

/* Configuration parameters of RFC6719. Reject parents that have a higher link metric than the following. The default
      value is 512 but we use 1024. */

#define MAX_LINK_METRIC 1024 /* Eq ETX of 8 */

/* Hysteresis of MRHOF: the rank must differ more than PARENT_SWITCH_THRESHOLD_DIV
 * in order to switch preferred parent. Default in RFC6719: 192, eq ETX of 1.5.
 * We use a more aggressive setting: 96, eq ETX of 0.75.
 */

#define PARENT_SWITCH_THRESHOLD 96 /* Eq ETX of 0.75 */

#else /* !RPL_MRHOF_SQUARED_ETX */

#define MAX_LINK_METRIC 2048 /* Eq ETX of 4 */

#define PARENT_SWITCH_THRESHOLD 160 /* Eq ETX of 1.25 (results in a churn comparable to the threshold of 96 in
      the non-squared case) */


#endif /* !RPL_MRHOF_SQUARED_ETX */

/* Reject parents that have a higher path cost than the following. */

#define MAX_PATH_COST 32768   /* Eq path ETX of 256 */
```

Figure 7.1: RPL Code for guiding parent selection

objective function F utilises a predefined metric. Different objective functions might be created in order to meet specific optimisation criteria for certain applications.

The *Minimum Rank with Hysteresis Objective Function* is the objective function that is used by default with RPL in Contiki-NG. MHROF selects routes that reduce additive routing parameters such as latency, hop count, and ETX. MHROF is, thus, intended to be extended with various metrics for calculating the cost of a path. The Expected Transmission Count (ETX) of links is a default metric defined by RFC 6719 and used in most RPL implementations to calculate the path with the lowest cost. With DODAGs constructed using MHROF and the ETX measure, nodes tend to choose parents with the best link quality and shortest hop-count to the root node (or sink).

In the context of this work, children serve as a P resource provider since they have messages to convey (i.e., the resource is the message). All prospective parents operate as resource requesters R in order to forward messages towards the sink. We view the children as trustworthy (since they are the source of the resource), whereas their potential parents may act arbitrarily.

In this situation, parents agree to maintaining a high-quality connection with their children, the same connection quality advertised to children prior to parent selection. Therefore, the resource requester transfers a message's aggregated quality metric. In this experiment, we deployed two mobile nodes, so making the quality metric dynamic and variable in response to their movements. Moreover, as mobile nodes migrate away from their parents, the quality metric declines. As a result, the mobile node will assume that the resource requester (its parent) is sending requests

Parameter	Value
Topology	Grid
Network Size	25 - 50
Routing	RPL
Mote Type	Z1
Simulation Duration	24h Cooja Time

Table 7.1: Contiki-NG Cooja Simulation Parameters RPL Mobility.

(or messages) that do not comply with the specification, i.e. it is an adversarial node.

More specifically, $\mathcal{SPEC}$ for a node n is as follows:

1. $\forall m \in neighbours(n), m$ is closer to the sink than n :
   $n.parent.quality \geq n.m.quality$

2. $n.parent \neq n.parent' \Rightarrow$
   $n.parent.quality \geq n.parent'.quality$
   for a determined number of rounds, parent' is the previous parent.

Please observe that this specification $\mathcal{SPEC}$ is fusion closed as the implementation of RPL contains history variables. Thus, RPL is $\mathcal{SPEC}$-auditable for a message m from node n. More information pertaining to the parent selection specification of RPL, the implementation of the $\mathcal{SPEC}$, can be found in Figures 7.3 and 7.1.

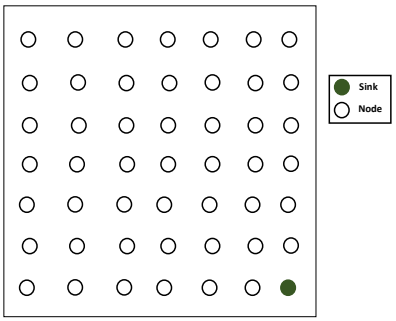


Figure 7.2: Grid Topology Used in the Simulations

We simulate the RPL protocol in Contiki-NG [2] using the Cooja simulator in a grid topology Figure 7.2. We set up a simple RPL UDP connection, with one server and multiple clients sending update messages regularly. We provided the node's id and the moving positions in the mobility plug-in. In total, we supplied 16 positions for two nodes that keep on repeating throughout the simulation.

With two versions of the simulation, one with a $5x5$ network and the other with a $7x7$ network, repeated a couple of times, we can see the performance in a network with two mobile nodes and a network with all the nodes stable in Figure 7.4.

```
int
rpl_process_parent_event (rpl_instance_t *instance, rpl_parent_t *p)
{
  int return_value;
  rpl_parent_t * last_parent = instance->current_dag->preferred_parent;

#if LOG_DBG_ENABLED
    rpl_rank_t old_rank;
    old_rank = instance->current_dag->rank;
#endif /* LOG_DBG_ENABLED */

  return_value = 1;

  if(RPL_IS_STORING(instance)
      && uip_ds6_route_is_nexthop(rpl_parent_get_ipaddr(p))
      && !rpl_parent_is_reachable(p) && instance->mop > RPL_MOP_NON_STORING) {
    LOG_WARN("Unacceptable link %u, removing routes via: ", rpl_get_parent_link_metric(p));
    LOG_WARN_6ADDR(rpl_parent_get_ipaddr(p));
    LOG_WARN_("\n");
    rpl_remove_routes_by_nexthop(rpl_parent_get_ipaddr(p), p->dag);
  }

  if(!acceptable_rank(p->dag, rpl_rank_via_parent(p))) {
    /* The candidate parent is no longer valid: the rank increase resulting
       from the choice of it as a parent would be too high. */
    LOG_WARN("Unacceptable rank %u (Current min %u, MaxRankInc %u)\n", (unsigned)p->rank,
        p->dag->min_rank, p->dag->instance->max_rankinc);
    rpl_nullify_parent(p);
    if(p != instance->current_dag->preferred_parent) {
      return 0;
    } else {
      return_value = 0;
    }
  }

  if(rpl_select_dag(instance, p) == NULL) {
    if(last_parent != NULL) {
      /* No suitable parent anymore; trigger a local repair. */
      LOG_ERR("No parents found in any DAG\n");
      rpl_local_repair(instance);
      return 0;
    }
  }

#if LOG_DBG_ENABLED
  if(DAG_RANK(old_rank, instance) != DAG_RANK(instance->current_dag->rank, instance)) {
    LOG_INFO("Moving in the instance from rank %hu to %hu\n",
      DAG_RANK(old_rank, instance), DAG_RANK(instance->current_dag->rank, instance));
    if(instance->current_dag->rank != RPL_INFINITE_RANK) {
      LOG_DBG("The preferred parent is ");
      LOG_DBG_6ADDR(rpl_parent_get_ipaddr(instance->current_dag->preferred_parent));
      LOG_DBG_(" (rank %u)\n",
          (unsigned)DAG_RANK(instance->current_dag->preferred_parent->rank, instance));
    } else {
      LOG_WARN("We don't have any parent");
    }
  }
#endif /* LOG_DBG_ENABLED */

  return return_value;
}
```

Figure 7.3: RPL Code for Guiding Parent Selection

In Figure 7.4 we see the packet delivery ratio of both scenarios, and we can observe that as the static network yields a 100% PDR, the network with mobile nodes yields a less optimal delivery ratio of 97%. While the difference is not much, it is clear that as a result of moving nodes and parent switches that do not move,

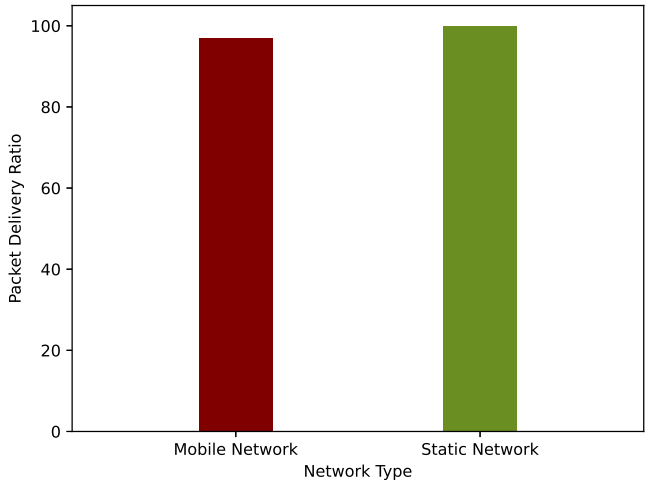


Figure 7.4: Packet Delivery Ratio Comparison Between the Static Network and the Mobile Network

there a percentage of message loss.

We can see further more significant results when we analyse other metrics such as the number of parent switches and the total energy consumption of the nodes.

- Parent Switches.
- Energy Consumption.

7.6.1 Parent Switches

RPL establishes the DODAG of the network based on the rank calculations. The rank in RPL is calculated using several metrics, ETX and the link quality. A node in a RPL network chooses a neighbouring node as its parent if it has an ETX less than all the other neighbours and a good link quality, in other words, the neighbour that has the better rank value.

If the ETX decreases and the link quality stays the same, thus resulting in a better rank, the node will keep this parent. If the ETX increases, even if the link quality is the same, the node will switch to another parent. The same happens with the link quality; if the ETX stays the same yet the link quality decreases resulting in a lower rank, this will lead the node to switch its own parent.

Figure 7.5 shows the number of parent switches in both networks. We can see that:

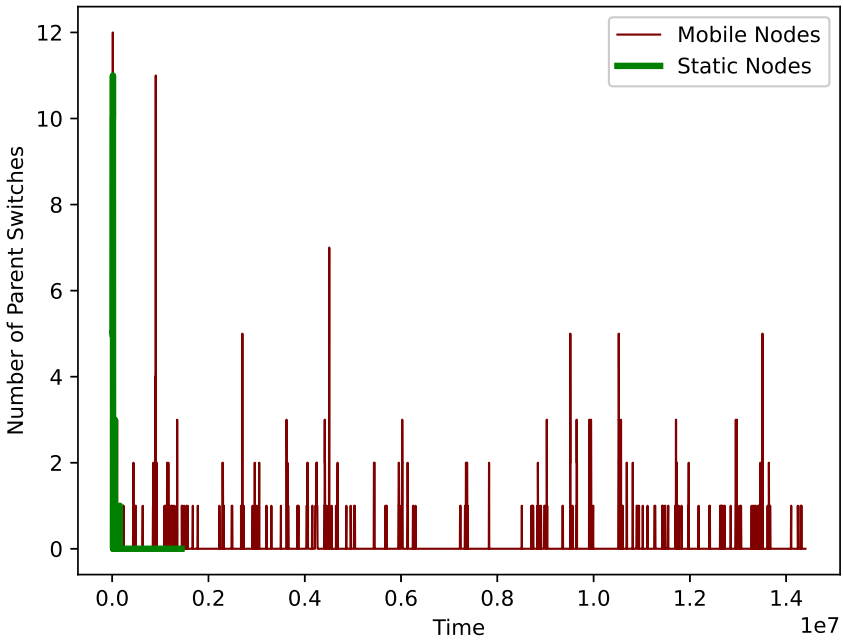


Figure 7.5: Parent Switches

1. The number of parent switches in the stable network is minimal and subtle.
2. The number of parent switches in the mobile network is more significant and frequent.

As the mobile nodes change positions in the mobile network, their relationship with their respective children changes. As they move and broadcast their metrics to rejoin the network, they will be picked out as parents for the neighbouring nodes.

Thus, committing to delivering the messages from the children to the sink, however, changing positions will increase the ETX violating the parent-child relation. Furthermore, it will also exhaust the network resources as RPL will keep repairing the DODAG.

In other words, as the parent position changes, the metrics change which means that the parent cannot provide the required resources for the child; thus the child will break the relationship and select another parent with better resources.

7.6.2 Energy Consumption

Energy consumption, as mentioned earlier, is an essential factor in keeping the network alive in IoTWS networks. Keeping the network stable will utilise the

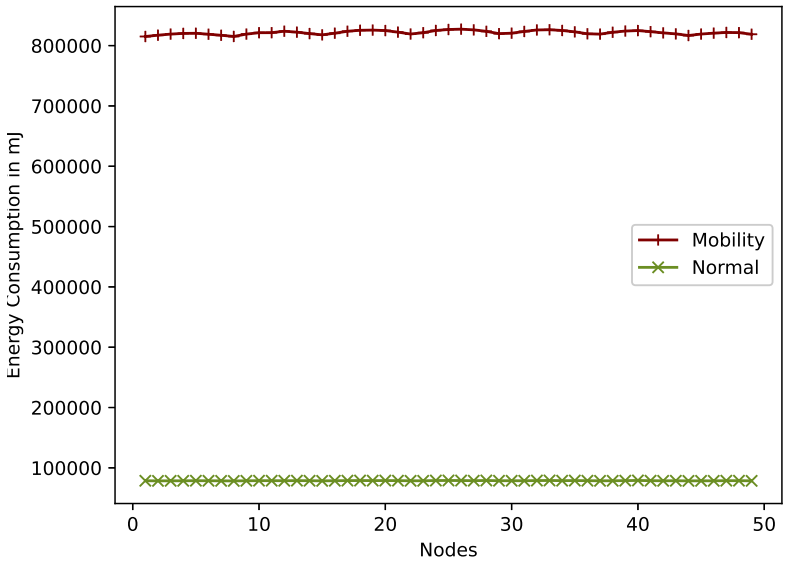


Figure 7.6: The Average Energy Consumption of All the Nodes in Both Mobile and Static Networks.

node's energy and keep consumption to a minimum. Thus designing protocols that minimise energy use such that the nodes stay alive as long as possible is essential. Figure 7.6 shows the average overall energy consumption of the nodes, and we can summarise the results as:

1. The energy consumption of the stable network is relatively low throughout the nodes in the network.

2. The energy consumption of the nodes in the mobile network is relatively high throughout the network.

The increased energy consumption of the nodes in the mobile networks shows that the movement did not only affect the neighbouring nodes of the mobile nodes; it affected the whole network.

This is because once a node switches its parent, the routing table will need to be updated from the root of the network; in this case, the server and the topology will also be updated, thus requiring the collaboration of the whole nodes in the network, which leads to the resources exhausting in terms of the nodes energy.

7.7 Conclusion

In this chapter, we examined the auditability problem in the Internet of Things wireless sensor-based networks. We defined the audibility problem and demonstrated that a system could become auditable if it satisfied the transparency property and could retain its execution history; in other words, it is a fusion closed system.

Without a trusted process, solving the strong auditability problem in an asynchronous system is impossible. We also demonstrated that the problem of strong auditability was as challenging to resolve as the fair exchange problem. Using a case study, we demonstrated that the RPL protocol was auditable by design, as it used its execution history throughout the protocol run time.

Chapter 8

Discussion

In the previous chapters, we sought to study whether it was possible to develop a middleware to log the network execution without interfering with its operation. We also investigated the possibility of this solution in the presence of Byzantine faults. We inquired about what constituted a correct log and what properties we needed to enforce to guarantee the log's completeness.

We can generate correct logs when we have a bounded number of Byzantine nodes, synchronous communications, and a connected network. We investigated the practicality of our middleware in several MAC settings. We discovered that it was difficult to achieve optimal middleware performance in the case of the TSCH MAC unless we added an extra slot in the TSCH scheduler of each node to deliver control messages.

We also found that doing so was only possible if we placed a limit on the number of Byzantine nodes in the network. We then established that through auditability, we could enforce logs specification, and we demonstrated that if the system's history was logged, the system could be transformed into an auditable system.

These findings are now discussed in this chapter. We evaluate the extent to which these findings address the research objectives and how they fit within the present body of knowledge. We will also describe the limits we encountered while conducting this research and the limitations imposed by our solutions.

To guide the conversation in a more organised manner, we will employ a Q&A format.

8.1 Assumptions

In this section, we discuss the assumptions we made throughout this research, whether they were explicit or implicit.

Q1: What role does the link reliability play in this study?

In our earlier research, we assumed that the links between nodes were reliable and messages would eventually be delivered, whether in a FIFO fashion or following numerous MAC layer retransmissions. The MAC layer would retransmit at least seven times before discarding the messages with this number of transmissions being configured; thus, it is fair to make such an assumption. This assumption is critical to our results.

Suppose no control message is sent when a data message is transmitted. In that case, it is impossible to determine whether the message was lost because of the reliability of the link or because the nodes are Byzantine and did not send any control messages. It is difficult to determine whether a data message was transmitted and lost owing to network unreliability or whether there was no data message sent and the nodes were forging control messages in the event of an orphan control message, which is a control message without a corresponding data message.

It is not unachievable, but it requires more detailed logging and middleware specifications to identify the status of links, which our algorithm in its current version is incapable of doing.

Q2: What if we cannot trust the loggers?

The loggers, a.k.a cluster heads, are the nodes that collect control messages from the nodes. Depending on the application, logs are either delivered to permanent storage immediately or retained for a limited time before being pushed to cloud storage. Regardless, the logger is responsible for gathering and preserving these logs until they are processed. In our work, we made the implicit assumption that the loggers could be trusted, which meant that they would not jeopardise the integrity of the logs in any way that raised doubts about whether the logs were correct.

If that assumption is missing, if we cannot trust the loggers to keep the logs safe, we cannot guarantee the logs' validity, hampering the log analysis findings.

The number of trusted processes in a system must be at least $3f+1$ to recover from any malicious behaviour; we used this conclusion to limit the number of faulty nodes in the network. Similar approaches may be applied to the logger, especially to ensure that, if we cannot trust the loggers, we have a sufficient number of correct loggers to allow the system to log its execution reliably.

Another alternative would be to employ a monitoring layer to isolate the Byzantine nodes from the clustering process, enabling only the correct nodes to be elected as loggers.

Q3: How can the 6top scheduling be modified to serve reliable logging?

We implemented 6top without modifying the negotiation rules or scheduling

function in our previous results. As previously stated, various scheduling functions are developed to meet industrial application needs. Our findings also revealed that it would only be possible to solve the logging problem once the schedulers allocated two slots per node every period.

Therefore, building a scheduling function for 6top that includes the control message slot allocation to its rules reduces the number of negotiations the nodes must perform. It also improves the likelihood of the nodes sending the control message and the logger receiving it.

***Q4: The impossibility results previously mentioned are related to the Strong Collection Problem* in the presence of Byzantine faults. Can these results relate to the case of mere crash faults?**

The impossibility results already mentioned in 7.3 indicated that it is impossible to solve the ***Strong Collection Problem*** in the presence of Byzantine faults unless we have G that is $L_i^{(2f+1)}$ connected, where L_i is cluster C_i logger.

However, in the case of crash faults, all the nodes are trusted; thus, to solve the ***Strong Collection Problem*** we only needed L_i to be $c+1$ connected, where c was the number of crashing nodes, and each node was equipped with a perfect crash detector.

As long as we have a *path* from node n to L_i , where at least one control message will be received by L_i, then we satisfy both the *Safety* and *Strong Termination* properties of ***Strong Collection Problem***.

***Q5: Can the Strong Collection Problem* be solved then with the presence of both crash and Byzantine faults?** As byzantine nodes can masquerade as crashing nodes, and we still have crashing nodes in the network, the cluster connectivity needs to be tightened up.

Thus for logger L_i in cluster C_i in G, we need L_i to be $2f + c + 1$ connected, where f is the number of malicious nodes, and c is the number of the crashed nodes.

8.2 Limitations

Every research will encounter a limitation at some point in its progression. While these limitations appear to impede progress, they allow for more innovative and adaptable solutions and allow the researcher to look into several potential solutions before finding the one that can work in their research. We understand this was the case in our research; while we could work around some constraints, overcoming the other constraints could be part of future efforts.

Q6: What are the constraints imposed as a result of the Internet of Things nodes memory limitations?

The nature of a node's memory is one of the most fundamental and challenging limitations that Internet of Things wireless sensor-based networks must contend with.

Nodes in IoT wireless sensor-based networks have $10KB$ of random access memory (RAM) and $100KB$ of read-only memory (ROM) [213]. Choosing what to save and keep and what to discard is always a matter of compromise; in our work, this did impose a limitation.

It was a limitation on multiple fronts, including the front of our algorithm's complexity. We could not permit the node to perform a more complex calculation, nor could we save logs in the node's buffer to be pushed later.

Another problem is when the routing protocol and message collisions are involved. Due to memory constraints, nodes had to discard messages when their buffers overflowed, which occurs when a logger simultaneously receives several control messages in the sender-oriented scenario. For the same reasons, nodes had to keep a limited routing table, decreasing the number of stored routes in order to function properly, and this contributes to the case message loss as well.

In the instance of TSCH MAC protocols, where memory was used to hold scheduler information, nodes were required to delete messages. In our previously reported results in 6.4, we had to reduce the network size in the test bed simulation in order to test RPL Classic with Orchestra and 6top, or we had to utilise RPL Lite to test our findings. In the first instance, the node's memory was incapable of forming the DAG due to all the other data it was required to store for TSCH schedulers, and in the second instance, nodes did not store the network routing table.

This is a typical issue in IoT networks; for example, the authors in [47] examine the memory constraint problem in depth from the standpoint of trust computation, providing an eviction technique that can select the least significant data to be removed from memory.

Q7: What are the loggers' high energy consumption ramifications?

In the CSMA and TSCH implementations, the loggers and their neighbouring nodes spend more energy than the other network nodes. As long as the loggers continue to receive control messages, it will be the responsibility of nearby nodes to pass these control messages to the logger.

There are several approaches to these challenges. One approach involves the clustering algorithm being configured to rotate the loggers depending on their energy, guaranteeing that they would not shut down over the network's lifespan. Another option would be to designate special nodes as the data loggers; these nodes would have greater energy and memory, enabling them to remain available for extended periods of time.

Q8: How can the auditability findings within an Internet of Things network embrace the trusted third-party premise?

Trusted third-party implementations are always concurrent with some underlying assumptions, such as connected networks and devices that have sufficient resources to support the implementation's overhead. In contrast, this is not the case in a network of things configuration, in which the network is only partially connected, the links are unstable, and the nodes have limited resources.

Nonetheless, the logger selection processes described before can be employed here. However, instead of the nodes acting as loggers, they will serve as trusted third parties in the network, only interacting with other nodes during transactions.

Regarding resource use, a similar argument may be made. If the TTP nodes' energy consumption is relatively high, specific network nodes might be designated as the TTP, or a rotation-based version of the clustering algorithm could be implemented.

8.3 Conclusion

This chapter discussed our findings and our limitations while conducting this research. We discussed the assumptions we made and why we made them. We also addressed how assumptions such as reliable links and trusted loggers could be handled in additional research.

We also investigated the limitations we faced and proposed ways to control and modify these limitations in our results by using various strategies drawn from the existing literature.

Chapter 9

Conclusion

In this final chapter, we will discuss the significance of our findings in light of our research questions and aims and their implications in the field at large. We also include looking back at the work's limitations and looking forward to its future potential.

9.1 Key Findings Summary

This study sought to solve the problem of reliable logging in the Internet of Things wireless sensor-based networks in the presence of Byzantine faults.

We defined the problem of logger selection and showed that it is an NP-Complete problem; we then defined the collection problem and presented several solutions to solve the problem with the presence of Byzantine faults. We then introduced the *Collector* algorithm and evaluated its performance in several simulations and testbed deployments, showing the algorithm's scalability. We studied and discussed the impact of different MAC platforms on our solutions and introduced a new variant of the collection problem for the case of TSCH MAC.

We discovered that an IoT-based wireless sensor network could be transformed into an auditable system if transparency and auditability requirements were met. We could also meet these requirements by adding history variables to the system. We demonstrated that this was unachievable in the absence of a trusted process and the circumstance of asynchronous communications. This was just as difficult to solve as the fair exchange problem.

9.2 Contribution to the field

Our main contribution is the formal definition of the *Collection Problem*. Our definition gives a comprehensive understanding of the logging problem within the

Internet of things wireless sensor-based network, and this understanding led to the development of the *Collector* algorithm, which offers one solution to this problem. According to our knowledge, this is the first study to tackle the logging problem from this angle.

We also present a new understanding of system auditability and how the use of history variables can transform systems into fusion closed systems and, as a result, auditable systems in an Internet of things wireless sensor-based network, laying the basis for understanding future auditability implementation in this field.

While the current state of the art gives technical specifics on logging for debugging, nothing has been done to merge the Byzantine model with logging inside a setup similar to the Internet of Things sensor wireless-based networks. Our study gives knowledge of how Byzantine behaviour influences the logging process. It offers insight into the trade-offs that must be made in order to tolerate discrepancies due to the Byzantine effects. In addition, this is one of the few works containing no cryptographic primitives.

We also evaluated the influence of various MAC configurations on the logging process and the Byzantine presence; we demonstrated that TSCH schedulers like 6TiSCH and Orchestra required more complete setups and better scheduling algorithms in order to be more Byzantine-aware.

Our approach to establishing auditability via the perspective of fair exchange is the first of its kind.

9.3 Limitations

As noted in the discussion chapter and as is the case with all research endeavours, this study has encountered some limitations. Since beginning our research in the area of the Internet of Things, we have experienced the same constraints as every other researcher in this area.

The restricted capabilities of the Internet of Things nodes in terms of memory, computation and energy constituted a significant limitation. As previously established, memory restrictions restrict the amount of memory our solution can consume before affecting network performance. On the other hand, there was always a limit on the energy consumption we could allow the nodes without reducing their network lifetime.

However, as we noted earlier, while we navigated these limitations as best we could throughout this study, in the future additional measures could be adopted to mitigate their consequences.

9.4 Future Work

This study establishes the foundation of the problem of reliably logging in the presence of Byzantine faults in the network; it also paves the way for many additional research lines that might build on this formalisation.

One such direction is enhancing the *Collector* algorithm to accommodate unreliable links. We could accomplish this in a number of approaches, including limiting cluster sizes and utilising RPL strong metrics to strengthen the connections between nodes.

A prominent direction is examining TSCH schedulers and how to reconstruct them so that they provide accurate logs in the presence of Byzantine faults. This can be accomplished, for instance, by adding rules to the slot assignment process to synchronise the nodes with their logger neighbours.

Specifically, the 6top layer offers an exciting direction. As 6top permits the nodes to dynamically negotiate the slot allocation based on the scheduling function's rules. Several design considerations could be made to improve the scheduling function and optimise the logging procedure. We can construct a scheduling function that assigns slots depending on the node logger and cluster neighbours. Another approach would be to design an intelligent scheduling function capable of predicting when nodes will transmit data messages and allocating slots accordingly so that each node has a slot to send control messages to the logger.

Investigating the case of auditable properties could be a future study project at work. This research examined how adding history variables can transform systems into auditable systems. Investigating a method to add a history variable to the properties would transform system properties into auditable ones, offering pointers for a new direction.

Clustering in RPL is another intriguing area of research. As previously mentioned, RPL enables clustering in which the cluster heads are prospective parents of the nodes or clustering upward. While in the case of simple data collection to the sink, this may appear trivial, as RPL is widely used in IoT applications and clustering is a solution for various network issues, it would be advantageous to develop a balanced approach that could support RPL non-based clustering.

Formalising the logging problem will provide a foundation for comprehending some challenges we face while creating dependable IoT networks. While these are the possibilities we can derive from this work, we can examine other IoT issues from a different angle and provide dependable solutions by understanding this problem.

In this thesis, we thoroughly investigated the *Collection Problem* in the Byzantine fault model. Based on our findings, incorporating crashing faults into the fault modules would be a valuable direction for future work.

9.5 Summary

In this chapter, we end the thesis's work. This thesis examined the complexity of reliable logging in the presence of Byzantine nodes; we presented several impossible results and evaluated the performance of our proposed solution in different MAC settings. Investigating the case of auditability in the Internet of Things wireless sensor-based network, we showed we could use logs and histories to transform systems into auditable ones.

We presented our findings and discussed how they fit into the existing body of research; we also reviewed the limitations of this study and its potential future implications.

Bibliography

[1] The Z1 mote · Zolertia/Resources Wiki — github.com. https://github.com/Zolertia/Resources/wiki/The-Z1-mote. [Accessed 27-Aug-2022].

[2] The open source operating system for the internet of things. URL http://www.contiki-os.org/.

[3] Platform cooja · contiki-ng/contiki-ng Wiki — github.com. https://github.com/contiki-ng/contiki-ng/wiki/Platform-cooja. [Accessed 27-Aug-2022].

[4] URL https://www.willow.co.uk/TelosB_Datasheet.pdf.

[5] Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(85)90056-0.

[6] Design and analysis of a fast local clustering service for wireless sensor networks. In *First International Conference on Broadband Networks*, pages 700–709. IEEE, 2004.

[7] Martin Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[8] Soroush Abbasian Dehkordi, Kamran Farajzadeh, Javad Rezazadeh, Reza Farahbakhsh, Kumbesan Sandrasegaran, and Masih Abbasian Dehkordi. A survey on data aggregation techniques in iot sensor networks. *Wireless Networks*, 26(2):1243–1263, 2020.

[9] Michael Abd-El-Malek, Gregory R Ganger, Garth R Goodson, Michael K Reiter, and Jay J Wylie. Fault-scalable byzantine fault-tolerant services. *ACM SIGOPS Operating Systems Review*, 39(5):59–74, 2005.

[10] Ghaida Muttashar Abdulsahib and Osamah Ibrahim Khalaf. An improved algorithm to fire detection in forest by using wireless sensor networks. *International Journal of Civil Engineering & Technology (IJCIET)-Scopus Indexed*, 9 (11):369–377, 2018.

[11] Arup Acharya and BR Badrinath. Recording distributed snopshots based on causal order of message delivery. *Information Processing Letters*, 44(6): 317–321, 1992.

[12] Ghaihab Hassan Adday, Shamala K Subramaniam, Zuriati Ahmad Zukarnain, and Normalia Samian. Fault tolerance structures in wireless sensor networks (wsns): Survey, classification, and future directions. *Sensors*, 22(16):6041, 2022.

[13] Cédric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frédéric Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. Fit iot-lab: A large scale open experimental iot testbed. Milan, Italy, December 2015. URL `https://hal.inria.fr/hal-01213938`.

[14] Kofi Sarpong Adu-Manu, Felicia Engmann, Godwin Sarfo-Kantanka, Godwill Enchill Baiden, and Bernice Akusika Dulemordzi. Wsn protocols and security challenges for environmental monitoring applications: A survey. *Journal of Sensors*, 2022, 2022.

[15] Amitanand S Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 45–58, 2005.

[16] Ian F Akyildiz, Tommaso Melodia, and Kaushik R Chowdhury. A survey on wireless multimedia sensor networks. *Computer networks*, 51(4):921–960, 2007.

[17] Alia Al Sadawi, Mohamed S Hassan, and Malick Ndiaye. A survey on the integration of blockchain with iot to enhance performance and eliminate challenges. *IEEE Access*, 9:54478–54497, 2021.

[18] Malak Alamri, NZ Jhanjhi, and Mamoona Humayun. Blockchain for internet of things (iot) research issues challenges & future directions: A review. *Int. J. Comput. Sci. Netw. Secur*, 19(1):244–258, 2019.

[19] Atif Ali, Yasir Khan Jadoon, Sabir Ali Changazi, and Muhammad Qasim. Military operations: Wireless sensor networks based applications to reinforce future battlefield command system. In *2020 IEEE 23rd International Multitopic Conference (INMIC)*, pages 1–6. IEEE, 2020.

[20] Mohammed H Alsharif, Sunghwan Kim, and Nuri Kuruoğlu. Energy harvesting techniques for wireless sensor networks/radio-frequency identification: A review. *Symmetry*, 11(7):865, 2019.

[21] Ahlam Saud Althobaiti and Manal Abdullah. Medium access control protocols for wireless sensor networks classifications and cross-layering. *Procedia Computer Science*, 65:4–16, 2015. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2015.09.070. URL `https://www.sciencedirect.com/science/article/pii/S1877050915029002`. International Conference on Communications, management, and Information technology (ICCMIT'2015).

[22] Raj Anwit and Prasanta K. Jana. *Proceedings of the 21st International Conference on Distributed Computing and Networking*. doi: 10.1145/3369740.3369769.

[23] Juan Aranda, Diego Mendez, and Henry Carrillo. Multimodal wireless sensor networks for monitoring applications: A review. *Journal of Circuits, Systems and Computers*, 29(02):2030003, 2020.

[24] Anish Arora and Sandeep S. Kulkarni. Detectors and correctors: A theory of fault-tolerance components. In *Proceedings of the 18th International Conference on Distributed Computing Systems, Amsterdam, The Netherlands, May 26-29, 1998*, pages 436–443. IEEE Computer Society, 1998. doi: 10.1109/ICDCS.1998.679772. URL `https://doi.org/10.1109/ICDCS.1998.679772`.

[25] Anish Arora and Sandeep S Kulkarni. Detectors and correctors: A theory of fault-tolerance components. In *Proceedings. 18th international conference on distributed computing systems (Cat. No. 98CB36183)*, pages 436–443. IEEE, 1998.

[26] Nadarajah Asokan. *Fairness in electronic commerce*. PhD thesis, University of Waterloo, 1998.

[27] Nadarajah Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 7–17, 1997.

[28] Nadarajah Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No. 98CB36186)*, pages 86–99. IEEE, 1998.

[29] Nadarajah Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in communications*, 18(4):593–610, 2000.

[30] Kenny Awuson-David, Tawfik Al-Hadhrami, Mamoun Alazab, Nazaraf Shah, and Andrii Shalaginov. Bcfl logging: An approach to acquire and preserve

admissible digital forensics evidence in cloud ecosystem. *Future Generation Computer Systems*, 122:1–13, 2021.

[31] Mohamed Saad Azizi and Moulay Lahcen Hasnaoui. An energy efficient clustering protocol for homogeneous and heterogeneous wireless sensor network. *ACM International Conference Proceeding Series*, Part F1481, 2019. doi: 10.1145/3320326.3320396.

[32] Matteo Baire, Andrea Melis, Matteo B Lodi, Pierluigi Tuveri, Chiara Dachena, Marco Simone, Alessandro Fanti, Giorgio Fumera, Tonino Pisanu, and Giuseppe Mazzarella. A wireless sensors network for monitoring the carasau bread manufacturing process. *Electronics*, 8(12):1541, 2019.

[33] BI Bakare and JD Enoch. A review of simulation techniques for some wireless communication system. *International Journal of Electronics Communication and Computer Engineering*, 10(2):60–70, 2019.

[34] Michel Bakni, Luis Manuel Moreno Chacón, Yudith Cardinale, Guillaume Terrasson, and Octavian Curea. Wsn simulators evaluation: an approach focusing on energy awareness. *arXiv preprint arXiv:2002.06246*, 2020.

[35] Nazreen Banu, Samia Souissi, Taisuke Izumi, and Koichi Wada. An improved byzantine agreement algorithm for synchronous systems with mobile faults. *International Journal of Computer Applications*, 43(22):1–7, 2012.

[36] Feng Bao, Robert H Deng, and Wenbo Mao. Efficient and practical fair exchange protocols with off-line ttp. In *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No. 98CB36186)*, pages 77–85. IEEE, 1998.

[37] Mohammed Sulaiman BenSaleh, Raoudha Saida, Yessine Hadj Kacem, and Mohamed Abid. Wireless sensor network design methodologies: A survey. *Journal of Sensors*, 2020, 2020.

[38] Bharat Bhushan and G Sahoo. Routing protocols in wireless sensor networks. In *Computational intelligence in sensor networks*, pages 215–248. Springer, 2019.

[39] Bharat Bhushan and Gadadhar Sahoo. Requirements, protocols, and security challenges in wireless sensor networks: An industrial perspective. In *Handbook of computer networks and cyber security*, pages 683–713. Springer, 2020.

[40] Kenneth P. Birman and Thomas A. Joseph. Exploiting virtual synchrony in distributed systems. *Proceedings of the 11th ACM Symposium on Operating*

Systems Principles, SOSP 1987, pages 123–138, 1987. doi: 10.1145/41457.37515.

[41] Bhaskar Biswas et al. Comparison of csma based mac protocols of wireless sensor networks. *arXiv preprint arXiv:1205.1701*, 2012.

[42] Johannes Blanckenstein, Jirka Klaue, and Holger Karl. A survey of low-power transceivers and their applications. *IEEE Circuits and Systems Magazine*, 15 (3):6–17, 2015. doi: 10.1109/MCAS.2015.2450634.

[43] Fatemeh Borran, Ravi Prakash, and André Schiper. Consensus in Wireless Ad Hoc Networks. (Infoscience):31–37, 2008.

[44] Yassine Boufenneche, Rafik Zitouni, Laurent George, and Nawel Gharbi. Network formation in 6tisch industrial internet of things under misbehaved nodes. In *2020 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 1–6. IEEE, 2020.

[45] Yassine Boufenneche, Rafik Zitouni, Laurent George, and Nawel Gharbi. Selfishness in secure internet of things networks: 6tisch case study. *Wireless Networks*, 27(6):3927–3946, 2021.

[46] Gabriel Bracha and Sam Toueg. Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985. ISSN 1557735X. doi: 10.1145/4221.214134.

[47] Matthew Bradbury, Arshad Jhumka, and Tim Watson. Information management for trust computation on resource-constrained iot devices. *Future Generation Computer Systems*, 135:348–363, 2022. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2022.05.004. URL `https://www.sciencedirect.com/science/article/pii/S0167739X22001698`.

[48] Peter Brucker. On the complexity of clustering problems. In *Optimization and Operations Research: Proceedings of a Workshop Held at the University of Bonn, October 2–8, 1977*, pages 45–54. Springer, 1978.

[49] MA Burhanuddin, Ali Abdul-Jabbar Mohammed, Ronizam Ismail, Mustafa Emad Hameed, Ali Noori Kareem, and Halizah Basiron. A review on security challenges and features in wireless sensor networks: Iot perspective. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 10(1-7):17–21, 2018.

[50] A. Burns, M. Nicholson, K. Tindell, and N. Zhang. Allocating and scheduling hard real-time tasks on a point-to-point distributed system. In *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*, pages 11–20, 1993.

[51] Carlene EA Campbell, Ibrar A Shah, and Kok-Keong Loo. Medium access control and transport protocol for wireless sensor networks: An overview. 2019.

[52] Matthias Carlier, Carlos M García Algora, An Braeken, and Kris Steenhaut. Analysis of internet protocol based multicast on duty-cycled wireless sensor networks. *IEEE Sensors Journal*, 18(10):4317–4327, 2018.

[53] Marco Carminati, Olfa Kanoun, Silvia L Ullo, and Salvo Marcuccio. Prospects of distributed wireless sensor networks for urban environmental monitoring. *IEEE Aerospace and Electronic Systems Magazine*, 34(6):44–52, 2019.

[54] Louie Chan, Karina Gomez Chavez, Heiko Rudolph, and Akram Hourani. Hierarchical routing protocols for wireless sensor network: A compressive survey. *Wireless Networks*, 26(5):3291–3314, 2020.

[55] K. Mani Chandy and Leslie Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transactions on Computer Systems (TOCS)*, 3(1):63–75, 1985. ISSN 15577333. doi: 10.1145/214451.214456.

[56] Tengfei Chang, Malisa Vucinic, Xavier Vilajosana, Simon Duquennoy, and Diego Dujovne. 6tisch minimal scheduling function (msf). *Internet Engineering Task Force, Internet-Draft draft-ietf-6tischmsf-02*, 2019.

[57] Sanjay Chaturvedi, Arun Parakh, and HK Verma. Development of a wsn based data logger for electrical power system. In *Information, Communication and Computing Technology: 4th International Conference, ICICCT 2019, New Delhi, India, May 11, 2019, Revised Selected Papers 4*, pages 72–83. Springer, 2019.

[58] An Ran Chen. An empirical study on leveraging logs for debugging production failures. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 126–128. IEEE, 2019.

[59] Shengbo Chen, Lanxue Zhang, Yuanmin Tang, Cong Shen, Roshan Kumar, Keping Yu, Usman Tariq, and Ali Kashif Bashir. Indoor temperature monitoring using wireless sensor networks: a smac application in smart cities. *Sustainable Cities and Society*, 61:102333, 2020.

[60] Long Cheng, Jianwei Niu, Chengwen Luo, Lei Shu, Linghe Kong, Zhiwei Zhao, and Yu Gu. Towards minimum-delay and energy-efficient flooding in low-duty-cycle wireless sensor networks. *Computer Networks*, 134:66–77, 2018.

[61] Edward Chuah, Arshad Jhumka, Sai Narasimhamurthy, John Hammond, James C. Browne, and Bill Barth. Linking resource usage anomalies with system failures from cluster log data. In *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*, pages 111–120, 2013. doi: 10.1109/SRDS.2013.20.

[62] Hao Chung, Elisaweta Masserova, Elaine Shi, and Sri AravindaKrishnan Thyagarajan. Ponyta: Foundations of side-contract-resilient fair exchange. *Cryptology ePrint Archive*, 2022.

[63] Cosmin Cirstea, Mihail Cernaianu, and Aurel Gontean. Packet loss analysis in wireless sensor networks routing protocols. In *2012 35th International Conference on Telecommunications and Signal Processing (TSP)*, pages 37–41, 2012. doi: 10.1109/TSP.2012.6256248.

[64] Alem Čolaković and Mesud Hadžialić. Internet of things (iot): A review of enabling technologies, challenges, and open research issues. *Computer networks*, 144:17–39, 2018.

[65] Moteiv Corporaton. Tmote sky: Datasheet, 2006. URL `http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf`.

[66] Miguel Correia, Giuliana Santos Veronese, Nuno Ferreira Neves, and Paulo Verissimo. Byzantine consensus in asynchronous message-passing systems: a survey. *International Journal of Critical Computer-Based Systems*, 2(2): 141–161, 2011.

[67] Khalid A Darabkh, Muna Al-Akhras, Jumana N Zomot, and Mohammed Atiquzzaman. Rpl routing protocol over iot: A comprehensive survey, recent advances, insights, bibliometric analysis, recommendations, and future directions. *Journal of Network and Computer Applications*, page 103476, 2022.

[68] Santosh Kumar Das, Sourav Samanta, Nilanjan Dey, and Rajesh Kumar. *Design frameworks for wireless networks*. Springer, 2020.

[69] Ian Davidson and SS Ravi. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *Knowledge Discovery in Databases: PKDD 2005: 9th European Conference on Principles and Practice*

of Knowledge Discovery in Databases, Porto, Portugal, October 3-7, 2005. Proceedings 9, pages 59–70. Springer, 2005.

[70] M. Demirbas, A Arora, V. Mittal, and Vinod Kulathumani. Design and analysis of a fast local clustering service for wireless sensor networks. pages 700– 709, 11 2004. ISBN 0-7695-2221-1. doi: 10.1109/BROADNETS.2004.30.

[71] Mrinai M. Dhanvijay and Shailaja C. Patil. Internet of things: A survey of enabling technologies in healthcare and its applications. *Computer Networks*, 153:113–131, 2019. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2019.03.006. URL `https://www.sciencedirect.com/science/article/pii/S1389128619302695`.

[72] Amadou Diarra, Sonia Ben Mokhtar, Pierre Louis Aublin, and Vivien Quéma. FullReview: Practical accountability in presence of selfish nodes. *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, 2014-Janua:271–280, 2014. ISSN 10609857. doi: 10.1109/SRDS.2014.32.

[73] E. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17, 01 1974.

[74] Danny Dolev, Michael J. Fischer, Rob Fowler, Nancy A. Lynch, and H. Raymond Strong. An efficient algorithm for byzantine agreement without authentication. *Information and Control*, 52(3):257–274, 1982. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(82)90776-8.

[75] Shlomi Dolev. *Self-stabilization*. MIT press, 2000.

[76] Kevin Driscoll, Brendan Hall, Håkan Sivencrona, and Phil Zumsteg. Byzantine fault tolerance, from theory to reality. In *International Conference on Computer Safety, Reliability, and Security*, pages 235–248. Springer, 2003.

[77] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. Orchestra: Robust mesh networks through autonomously scheduled tsch. In *Proceedings of the 13th ACM conference on embedded networked sensor systems*, pages 337–350, 2015.

[78] Simon Duquennoy, Atis Elsts, Beshr Al Nahas, and George Oikonomo. Tsch and 6tisch for contiki: Challenges, design and evaluation. In *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 11–18. IEEE, 2017.

[79] Simon Duquennoy, Joakim Eriksson, and Thiemo Voigt. Five-nines reliable downward routing in rpl, 2017. URL `https://arxiv.org/abs/1710.02324`.

[80] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 967–984, 2018.

[81] Eyman Fathelrhman Ahmed Elsmany, Mohd Adib Omar, Tat-Chee Wan, and Altahir Abdalla Altahir. Eesra: Energy efficient scalable routing algorithm for wireless sensor networks. *IEEE Access*, 7:96974–96983, 2019.

[82] Atis Elsts, Seohyang Kim, Hyung-Sin Kim, and Chongkwon Kim. An empirical survey of autonomous scheduling methods for tsch. *IEEE Access*, 8:67147–67165, 2020.

[83] P. D. Ezhilchelvan and S. K. Shrivastava. A family of trusted third party based fair-exchange protocols. *IEEE Transactions on Dependable and Secure Computing*, 2(4):273–286, Oct 2005. ISSN 1545-5971. doi: 10.1109/TDSC.2005.40.

[84] Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. Virtual synchrony guarantees for cyber-physical systems. *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, pages 20–30, 2013. ISSN 10609857. doi: 10.1109/SRDS.2013.11.

[85] Michael Fischer, Nancy Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32:374–382, 04 1985. doi: 10.1145/3149.214121.

[86] Reinaldo Padilha França, Yuzo Iano, Ana Carolina Borges Monteiro, and Rangel Arthur. Intelligent applications of wsn in the world: a technological and literary background. In *Handbook of Wireless Sensor Networks: Issues and Challenges in Current Scenario's*, pages 13–34. Springer, 2020.

[87] Stephen D Gantz. *The basics of IT audit: purposes, processes, and practical information*. Elsevier, 2013.

[88] JJ Garcia-Luna-Aceves. Improving carrier-sense multiple access using cues of channel utilization. In *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 190–198. IEEE, 2019.

[89] Michael R Garey and David S Johnson. "strong"np-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3): 499–508, 1978.

[90] Vassos Hadzilacos and Sam Toueg. Distributed systems (2nd ed.). chapter Fault-tolerant Broadcasts and Related Problems, pages 97–145. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993. ISBN 0-201-62427-3. URL `http://dl.acm.org/citation.cfm?id=302430.302435`.

[91] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. *Operating Systems Review (ACM)*, pages 175–188, 2007. ISSN 01635980. doi: 10.1145/1294261.1294279.

[92] Yasir Babiker Hamdan et al. Smart home environment future challenges and issues-a survey. *Journal of Electronics*, 3(01):239–246, 2021.

[93] Xu Hao, Long Yu, Liu Zhiqiang, Liu Zhen, and Gu Dawu. Dynamic practical byzantine fault tolerance. In *2018 IEEE conference on communications and network security (CNS)*, pages 1–8. IEEE, 2018.

[94] Mohammed Zaki Hasan, Fadi Al-Turjman, and Hussain Al-Rizzo. Analysis of cross-layer design of quality-of-service forward geographic wireless sensor network routing strategies in green internet of things. *IEEE Access*, 6:20371–20389, 2018.

[95] Wendi B Heinzelman, Anantha P Chandrakasan, and Hari Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on wireless communications*, 1(4):660–670, 2002.

[96] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd annual Hawaii international conference on system sciences*, pages 10–pp. IEEE, 2000.

[97] Rodrigo Teles Hermeto, Antoine Gallais, and Fabrice Theoleyre. Scheduling for ieee802. 15.4-tsch and slow channel hopping mac in low power industrial wireless networks: A survey. *Computer Communications*, 114:84–105, 2017.

[98] Veena Hingarh and Arif Ahmed. *Understanding and conducting information systems auditing*. John Wiley & Sons, 2013.

[99] H Hoang. Csma based medium access control for wireless sensor network.

[100] Li Hong-Tan, Kong Cui-Hua, BalaAnand Muthu, and CB Sivaparthipan. Big data and ambient intelligence in iot-based wireless student health monitoring system. *Aggression and Violent Behavior*, page 101601, 2021.

[101] Wen-Wen Huang, Ya-Li Peng, Jian Wen, and Min Yu. Energy-efficient multi-hop hierarchical routing protocol for wireless sensor networks. In *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, volume 2, pages 469–472, 2009. doi: 10.1109/NSWCTC.2009.352.

[102] Christiana Ioannou and Vasos Vassiliou. An intrusion detection system for constrained wsn and iot nodes based on binary logistic regression. In *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 259–263, 2018.

[103] Valerie Issarny, Mauro Caporuscio, and Nikolaos Georgantas. A perspective on the future of middleware-based software engineering. *Future of Software Engineering (FOSE'07)*, pages 244–258, 2007.

[104] Sohail Jabbar, Abid Ali Minhas, Muhammad Imran, Shehzad Khalid, and Kashif Saleem. Energy efficient strategy for throughput improvement in wireless sensor networks. *Sensors*, 15(2):2473–2495, 2015.

[105] Richa Jain. Comparative analysis of contention based and tdma based mac protocols for wireless sensor networks. *International Journal of Information Technology*, 12(1):245–250, 2020.

[106] Seungbeom Jeong, Hyung-Sin Kim, Jeongyeup Paek, and Saewoong Bahk. Ost: On-demand tsch scheduling with traffic-awareness. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 69–78, 2020. doi: 10.1109/INFOCOM41043.2020.9155496.

[107] Qiangfeng Jiang, Yi Luo, and D Manivannan. An optimistic checkpointing and message logging approach for consistent global checkpoint collection in distributed systems. *Journal of Parallel and Distributed Computing*, 68(12): 1575–1589, 2008.

[108] L. Jiqiang, H. Zhen, and L. Zengwei. Secure audit logs server to support computer forensics in criminal investigations. In *2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering. TENCOM '02. Proceedings.*, volume 1, pages 180–183 vol.1, Oct 2002. doi: 10.1109/TENCON.2002.1181244.

[109] Pushpam Aji John, Rudolf Agren, Yu-Jung Chen, Christian Rohner, and Edith Ngai. 868 mhz wireless sensor network-a study. *arXiv preprint arXiv:1609.00475*, 2016.

[110] David B Johnson and Willy Zwaenepoel. Recovery in distributed systems using asynchronous message logging and checkpointing. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, pages 171–181, 1988.

[111] David Bruce Johnson. *Distributed system fault tolerance using message logging and checkpointing*. Rice University, 1990.

[112] Gyanendra Prasad Joshi, Seung Yeob Nam, and Sung Won Kim. Cognitive radio wireless sensor networks: Applications, challenges and research trends. *Sensors*, 13(9):11196–11228, 2013. ISSN 1424-8220. doi: 10.3390/s130911196. URL `https://www.mdpi.com/1424-8220/13/9/11196`.

[113] Alakesh Kalita and Manas Khatua. 6tisch–ipv6 enabled open stack iot network formation: A review. *ACM Transactions on Internet of Things*, 2022.

[114] Alakesh Kalita, Alessandro Brighente, Manas Khatua, and Mauro Conti. Effect of dis attack on 6tisch network formation. *IEEE Communications Letters*, 26 (5):1190–1193, 2022.

[115] Saeed Karimi-Bidhendi, Jun Guo, and Hamid Jafarkhani. Energy-efficient node deployment in heterogeneous two-tier wireless sensor networks with limited communication range. *IEEE Transactions on Wireless Communications*, 20 (1):40–55, 2020.

[116] Mandeep Kaur and Amit Munjal. Data aggregation algorithms for wireless sensor network: a review. *Ad Hoc Networks*, 100:102083, 2020.

[117] Harith Kharrufa, Hayder AA Al-Kashoash, and Andrew H Kemp. Rpl-based routing protocols in iot applications: a review. *IEEE Sensors Journal*, 19(15): 5952–5967, 2019.

[118] Nesrine Khelifi, Wafa Kammoun, and Habib Youssef. Efficiency of the rpl repair mechanisms for low power and lossy networks. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 98–103. IEEE, 2014.

[119] Seohyang Kim, Hyung-Sin Kim, and Chongkwon Kim. Alice: Autonomous link-based cell scheduling for tsch. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, pages 121–132, 2019.

[120] David Kozhaya, Jérémie Decouchant, and Paulo Esteves-Verissimo. Rt-byzcast: Byzantine-resilient real-time reliable broadcast. *IEEE Transactions on Computers*, 68(3):440–454, 2018.

[121] Kurtis Kredo II and Prasant Mohapatra. Medium access control in wireless sensor networks. *Computer networks*, 51(4):961–994, 2007.

[122] Mirko Křivánek and Jaroslav Morávek. Np-hard problems in hierarchical-tree clustering. *Acta informatica*, 23:311–323, 1986.

[123] Kubernetes. Cluster-level logging architectures. URL `https://kubernetes.io/docs/concepts/cluster-administration/logging/`.

[124] Pawan Kumar and SRN Reddy. Wireless sensor networks: A review of motes, wireless technologies, routing algorithms and static deployment strategies for agriculture applications. *CSI Transactions on ICT*, 8(3):331–345, 2020.

[125] Agus Kurniawan. *Practical Contiki-NG: programming for wireless sensor networks*. Apress, 2018.

[126] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*, pages 179–196. 2019.

[127] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pages 203–226. 2019.

[128] Carlos Andres Lara-Nino, Miguel Morales-Sandoval, and Arturo Diaz-Perez. Post-quantum cryptography on wireless sensor networks: Challenges and opportunities. *Integration of WSNs into Internet of Things*, pages 81–99, 2021.

[129] Alexandru Lavric and Valentin Popa. Performance evaluation of lorawan communication scalability in large-scale wireless sensor networks. *Wireless Communications and Mobile Computing*, 2018, 2018.

[130] Philip Levis, Thomas Clausen, Jonathan Hui, Omprakash Gnawali, and Jeong-Gil Ko. The trickle algorithm. Technical report, 2011.

[131] Chunfeng Liu, Zhao Zhao, Wenyu Qu, Tie Qiu, and Arun Kumar Sangaiah. A distributed node deployment algorithm for underwater wireless sensor networks based on virtual forces. *Journal of Systems Architecture*, 97:9–19, 2019.

[132] Jian Liu, Wenting Li, Ghassan O Karame, and N Asokan. Toward fairness of cryptocurrency payments. *IEEE Security & Privacy*, 16(3):81–89, 2018.

[133] Xiaowu Liu, Jiguo Yu, Feng Li, Weifeng Lv, Yinglong Wang, and Xiuzhen Cheng. Data aggregation in wireless sensor networks: from the perspective of security. *IEEE Internet of Things Journal*, 7(7):6495–6513, 2019.

[134] Xuxun Liu, Mohammad S Obaidat, Chi Lin, Tian Wang, and Anfeng Liu. Movement-based solutions to energy limitation in wireless sensor networks: State of the art and future trends. *iEEE Network*, 35(2):188–193, 2020.

[135] Y. Liu, K. Liu, and M. Li. Passive diagnosis for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 18(4):1132–1144, Aug 2010. doi: 10.1109/TNET.2009.2037497.

[136] Ying Liu, Kwan-Wu Chin, Changlin Yang, and Tengjiao He. Nodes deployment for coverage in rechargeable wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 68(6):6064–6073, 2019.

[137] Nicolas López, Cesar Azurdia-Meza, Claudio Valencia, and Samuel Montejo-Sánchez. On the performance of 6lowpan using tsch/orchestra mode against a jamming attack. In *2019 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pages 1–5. IEEE, 2019.

[138] Teresa F Lunt. Automated audit trail analysis and intrusion detection: A survey. In *Proceedings of the 11th National Computer Security Conference*, volume 353. Baltimore, MD, 1988.

[139] Michele Luvisotto, Zhibo Pang, and Dacfey Dzung. Ultra high performance wireless control for critical applications: Challenges and directions. *IEEE Transactions on Industrial Informatics*, 13(3):1448–1459, 2017. doi: 10.1109/TII.2016.2617459.

[140] Jemish Maisuria and Saurabh Mehta. An overview of medium access control protocols for cognitive radio sensor networks. *Multidisciplinary Digital Publishing Institute Proceedings*, 2(3):135, 2018.

[141] Sheunesu M Makura, HS Venter, Richard Adeyemi Ikuesan, Victor R Kebande, and Nickson M Karie. Proactive forensics: keystroke logging from the cloud as potential digital evidence for forensic readiness purposes. In *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, pages 200–205. IEEE, 2020.

[142] Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1041–1053, 2019.

[143] Shwetank Dattatraya Mamdiwar, Zainab Shakruwala, Utkarsh Chadha, Kathiravan Srinivasan, and Chuan-Yu Chang. Recent advances on iot-assisted wearable sensor systems for healthcare monitoring. *Biosensors*, 11(10):372, 2021.

[144] P. Massonet, S. Naqvi, C. Ponsard, J. Latanicki, B. Rochwerger, and M. Villari. A monitoring and audit logging architecture for data location compliance in federated cloud infrastructures. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 1510–1517, May 2011. doi: 10.1109/IPDPS.2011.304.

[145] Tales Benigno Matos, Angelo Brayner, and Jose Everardo Bessa Maia. Towards in-network data prediction in wireless sensor networks. *Proceedings of the ACM Symposium on Applied Computing*, pages 592–596, 2010. doi: 10.1145/1774088.1774210.

[146] Alexandre Maurer. Reliable Communication in a Dynamic Network in the Presence of Byzantine Faults. pages 1–14, 2018.

[147] Gregory Maxwell. Zero knowledge contingent payment. *Bitcoin Wiki*, 2011.

[148] Deep Medhi and Karthikeyan Ramasamy. Routing protocols: Framework and principles. 2018.

[149] Amira Meharouech, Jocelyne Elias, and Ahmed Mehaoua. Moving towards body-to-body sensor networks for ubiquitous applications: A survey. *Journal of sensor and Actuator Networks*, 8(2):27, 2019.

[150] Hitesh Mohapatra and Amiya Kumar Rath. Fault-tolerant mechanism for wireless sensor network. *IET wireless sensor systems*, 10(1):23–30, 2020.

[151] Sonia Ben Mokhtar. *Semantic middleware for service-oriented pervasive computing*. PhD thesis, Université Pierre et Marie Curie-Paris VI, 2007.

[152] Seyyed Keyvan Mousavi, Ali Ghaffari, Sina Besharat, and Hamed Afshari. Security of internet of things based on cryptographic algorithms: a survey. *Wireless Networks*, 27(2):1515–1555, 2021.

[153] Mikhail Nesterenko and Sebastien Tixeuil. Discovering network topology in the presence of byzantine faults. *IEEE Transactions on Parallel and Distributed Systems*, 20(12):1777–1789, 2009. ISSN 10459219. doi: 10.1109/TPDS.2009.25.

[154] Francisco Neves, Nuno Machado, and José Pereira. Falcon: A practical Log-Based analysis tool for distributed systems. *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018*, pages 534–541, 2018. doi: 10.1109/DSN.2018.00061.

[155] WM Nooriman, AH Abdullah, N Abdul Rahim, and K Kamarudin. Development of wireless sensor network for harumanis mango orchard's temperature, humidity and soil moisture monitoring. In *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pages 263–268. IEEE, 2018.

[156] George Oikonomou, Simon Duquennoy, Atis Elsts, Joakim Eriksson, Yasuyuki Tanaka, and Nicolas Tsiftes. The Contiki-NG open source operating system for next generation IoT devices. *SoftwareX*, 18:101089, 2022. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2022.101089.

[157] Henning Pagnia and Felix C Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Department of Computer Science, Darmstadt, Germany, 1999.

[158] Krerk Piromsopa and Richard J Enbody. Buffer-overflow protection: the theory. In *2006 IEEE International Conference on Electro/Information Technology*, pages 454–458. IEEE, 2006.

[159] Rahul Priyadarshi, Bharat Gupta, and Amulya Anurag. Deployment techniques in wireless sensor networks: a survey, classification, challenges, and future research issues. *The Journal of Supercomputing*, 76(9):7333–7373, 2020.

[160] Shazirawati Mohd Puzi, Shaharuddin Salleh, Ruzana Ishak, and Stephan Olariu. Neighborhood discovery in a wireless sensor networks. *ACM International Conference Proceeding Series*, pages 80–86, 2011. doi: 10.1145/2095697.2095713.

[161] Wei Qi, Wei Liu, Xuxun Liu, Anfeng Liu, Tian Wang, Neal N Xiong, and Zhiping Cai. Minimizing delay and transmission times with long lifetime in code dissemination scheme for high loss ratio and low duty cycle wireless sensor networks. *Sensors*, 18(10):3516, 2018.

[162] Ping Qian, Yang Songwei, Zhang Yong, Jin Hai, and Luo Zhiyuan. A Wireless Sensor Network Clustering Algorithm for Smart Grid Service Demand. *ACM*

International Conference Proceeding Series, pages 163–167, 2019. doi: 10.1145/3375998.3376036.

[163] Anagha Rajput and Vinoth Babu Kumaravelu. Scalable and sustainable wireless sensor networks for agricultural application of internet of things using fuzzy c-means algorithm. *Sustainable Computing: Informatics and Systems*, 22:62–74, 2019.

[164] Vidya Rao and KV Prema. A review on lightweight cryptography for internet-of-things based applications. *Journal of Ambient Intelligence and Humanized Computing*, 12:8835–8857, 2021.

[165] Manisha Rathee, Sushil Kumar, Amir H Gandomi, Kumar Dilip, Balamurugan Balusamy, and Rizwan Patan. Ant colony optimization based quality of service aware energy balancing secure routing algorithm for wireless sensor networks. *IEEE Transactions on Engineering Management*, 68(1):170–182, 2019.

[166] D Devi Kala Rathinam, D Surendran, A Shilpa, A Santhiya Grace, and J Sherin. Modern agriculture using wireless sensor network (wsn). In *2019 5th international conference on advanced computing & communication Systems (ICACCS)*, pages 515–519. IEEE, 2019.

[167] P.P. Ray. A survey on internet of things architectures. *Journal of King Saud University - Computer and Information Sciences*, 30(3):291–319, 2018. ISSN 1319-1578. doi: https://doi.org/10.1016/j.jksuci.2016.10.003.

[168] Muhammad Noman Riaz, Attaullah Buriro, and Athar Mahboob. Classification of attacks on wireless sensor networks: A survey. *International Journal of Wireless and Microwave Technologies*, 8(6):15–39, 2018.

[169] CH Sandeep et al. Security challenges and issues of the iot system. *Indian Journal of Public Health Research & Development*, 9(11), 2018.

[170] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000. doi: 10.1145/353323.353382. URL `https://doi.org/10.1145/353323.353382`.

[171] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2(2):159–176, May 1999. ISSN 1094-9224. doi: 10.1145/317087.317089. URL `http://doi.acm.org/10.1145/317087.317089`.

[172] Anuj Sehgal, Anthéa Mayzaud, Rémi Badonnel, Isabelle Chrisment, and Jürgen Schönwälder. Addressing dodag inconsistency attacks in rpl networks. In *2014 Global Information Infrastructure and Networking Symposium (GIIS)*, pages 1–8. IEEE, 2014.

[173] Munuswamy Selvi, K Thangaramya, Sannasi Ganapathy, Kanagasabai Kulothungan, H Khannah Nehemiah, and Arputharaj Kannan. An energy aware trust based secure routing algorithm for effective communication in wireless sensor networks. *Wireless Personal Communications*, 105(4):1475–1490, 2019.

[174] Ankit Shah and Margi Engineer. A survey of lightweight cryptographic algorithms for iot-based applications. In *Smart innovations in communication and computational sciences*, pages 283–293. Springer, 2019.

[175] Amin Shahraki, Amir Taherkordi, Øystein Haugen, and Frank Eliassen. Clustering objectives in wireless sensor networks: A survey and research direction analysis. *Computer Networks*, 180(March), 2020. ISSN 13891286. doi: 10.1016/j.comnet.2020.107376.

[176] Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

[177] Ashutosh Sharma, Rajiv Kumar, and Pawandeep Kaur. Study of issues and challenges of different routing protocols in wireless sensor network. In *2019 Fifth international Conference on image information processing (ICIIP)*, pages 585–590. IEEE, 2019.

[178] Ravi Sharma, Shiva Prakash, and Pankaj Roy. Methodology, applications, and challenges of wsn-iot. In *2020 International Conference on Electrical and Electronics Engineering (ICE3)*, pages 502–507. IEEE, 2020.

[179] Liqi Shi and Abraham O Fapojuwo. Tdma scheduling with optimized energy efficiency and minimum delay in clustered wireless sensor networks. *IEEE Transactions on Mobile Computing*, 9(7):927–940, 2010.

[180] M Shyama and Anju S Pillai. Fault-tolerant techniques for wireless sensor network—a comprehensive survey. *Innovations in Electronics and Communication Engineering*, pages 261–269, 2019.

[181] Abraham Silberschatz, Peter B Galvin, and Greg Gagne. Operating system concepts, edition 9, illustrated, 2009.

[182] Manish Kumar Singh, Syed Intekhab Amin, Syed Akhtar Imam, Vibhav Kumar Sachan, and Amit Choudhary. A survey of wireless sensor network and its types. In *2018 international conference on advances in computing, communication control and networking (ICACCCN)*, pages 326–330. IEEE, 2018.

[183] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. Industrial internet of things: Challenges, opportunities, and directions. *IEEE transactions on industrial informatics*, 14(11):4724–4734, 2018.

[184] Diomidis Spinellis. Debuggers and logging frameworks. *IEEE software*, 23(3): 98–99, 2006.

[185] Dan Steingart. Power sources for wireless sensor networks. In *Energy harvesting technologies*, pages 267–286. Springer, 2009.

[186] Anand Prabhu Subramanian and Samir R Das. Addressing deafness and hidden terminal problem in directional antenna based wireless multi-hop networks. *Wireless networks*, 16(6):1557–1567, 2010.

[187] MR Sumalatha and Pranab Batsa. Data collection and audit logs of digital forensics in cloud. In *Recent Trends in Information Technology (ICRTIT), 2016 International Conference on*, pages 1–8. IEEE, 2016.

[188] Yad Tahir, Shusen Yang, and Julie McCann. BRPL: Backpressure RPL for High-Throughput and Mobile IoTs. *IEEE Transactions on Mobile Computing*, 17(1):29–43, 2017. ISSN 1536-1233. doi: 10.1109/tmc.2017.2705680.

[189] Matthew Tancreti, Vinaitheerthan Sundaram, Saurabh Bagchi, and Patrick Eugster. TARDIS: Software-only system-level record and replay in Wireless Sensor Networks. *IPSN 2015 - Proceedings of the 14th International Symposium on Information Processing in Sensor Networks (Part of CPS Week)*, pages 286–297, 2015. doi: 10.1145/2737095.2737096.

[190] Henry S Teng, Kaihu Chen, and Stephen C Lu. Security audit trail analysis using inductively generated predictive rules. In *Artificial Intelligence Applications, 1990., Sixth Conference on*, pages 24–29. IEEE, 1990.

[191] K Thangaramya, Kanagasabai Kulothungan, R Logambigai, Munuswamy Selvi, Sannasi Ganapathy, and Arputharaj Kannan. Energy aware cluster and neuro-fuzzy based routing algorithm for wireless sensor networks in iot. *Computer Networks*, 151:211–223, 2019.

[192] Pascal Thubert, Tim Winter, Anders Brandt, Jonathan Hui, Richard Kelsey, Phil Levis, Kristofer Pister, Rene Struik, Jp Vasseur, and Roger Alexander. Rpl: Ipv6 routing protocol for low power and lossy networks. *IETF*, RFC 6550, 03 2012.

[193] Haseeb Touqeer, Shakir Zaman, Rashid Amin, Mudassar Hussain, Fadi Al-Turjman, and Muhammad Bilal. Smart home security: challenges, issues and solutions at different iot layers. *The Journal of Supercomputing*, 77(12): 14053–14089, 2021.

[194] Md Ashraf Uddin, Andrew Stranieri, Iqbal Gondal, and Venki Balasubramanian. A survey on the adoption of blockchain in iot: Challenges and solutions. *Blockchain: Research and Applications*, 2(2):100006, 2021.

[195] Jeffrey D. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[196] Andreas Ramstad Urke, Øivind Kure, and Knut Øvsthus. A survey of 802.15.4 tsch schedulers for a standardized industrial internet of things. *Sensors*, 22 (1), 2022. ISSN 1424-8220. doi: 10.3390/s22010015. URL `https://www.mdpi.com/1424-8220/22/1/15`.

[197] Tim Van Der Lee, Georgios Exarchakos, and Antonio Liotta. Distributed tsch scheduling: A comparative analysis. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3517–3522. IEEE, 2017.

[198] Xavier Vilajosana, Thomas Watteyne, Tengfei Chang, Mališa Vučinić, Simon Duquennoy, and Pascal Thubert. Ietf 6tisch: A tutorial. *IEEE Communications Surveys & Tutorials*, 22(1):595–615, 2019.

[199] Xavier Vilajosana, Thomas Watteyne, Mališa Vučinić, Tengfei Chang, and Kristofer SJ Pister. 6tisch: Industrial performance for ipv6 internet-of-things networks. *Proceedings of the IEEE*, 107(6):1153–1165, 2019.

[200] H. Vogt, H. Pagnia, and F. C. Gartner. Modular fair exchange protocols for electronic commerce. In *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*, pages 3–11, Dec 1999. doi: 10.1109/CSAC.1999.816008.

[201] Jin Wang, Yu Gao, Wei Liu, Arun Kumar Sangaiah, and Hye-Jin Kim. Energy efficient routing algorithm with mobile sink support for wireless sensor networks. *Sensors*, 19(7):1494, 2019.

[202] Qin Wang, Xavier Vilajosana, and Thomas Watteyne. 6tisch operation sublayer (6top) protocol (6p). Technical report, 2018.

[203] Yi-Min Wang and W Kent Fuchs. Optimistic message logging for independent checkpointing in message-passing systems. Technical report, ILLINOIS UNIV AT URBANA COORDINATED SCIENCE LAB, 1992.

[204] Xie, Haomeng Yan, Zheng, Atiquzzaman, and Mohammed. Data collection for security measurement in wireless sensor networks: A survey. *IEEE Internet of Things Journal*, (2):2205–2224, 2019. doi: 10.1109/JIOT.2018.2883403.

[205] Zheng Xie, Haomeng and Mohammed Yao, Zhen Atiquzzaman. Data collection for security measurement in wireless sensor networks: A survey. *IEEE Internet of Things Journal*, 6(2):2205–2224, 2018.

[206] Jinyuan Xu, Baoxing Gu, and Guangzhao Tian. Review of agricultural iot technology. *Artificial Intelligence in Agriculture*, 2022.

[207] Guang Yang, Lie Dai, and Zhiqiang Wei. Challenges, threats, security issues and new trends of underwater wireless sensor networks. *Sensors*, 18(11):3907, 2018.

[208] Ossama Younis and Sonia Fahmy. Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In *IEEE INFOCOM 2004*, volume 1. IEEE, 2004.

[209] Umar Zaman, Faisal Mehmood, Naeem Iqbal, Jungsuk Kim, and Muhammad Ibrahim. Towards secure and intelligent internet of health things: A survey of enabling technologies and applications. *Electronics*, 11(12):1893, 2022.

[210] Mahdi Zareei, AKM Muzahidul Islam, Cesar Vargas-Rosales, Nafees Mansoor, Shidrokh Goudarzi, and Mubashir Husain Rehmani. Mobility-aware medium access control protocols for wireless sensor networks: A survey. *Journal of Network and Computer Applications*, 104:21–37, 2018.

[211] Ming Zhao, Arun Kumar, Peter Han Joo Chong, and Rongxing Lu. A comprehensive study of rpl and p2p-rpl routing protocols: Implementation, challenges and opportunities. *Peer-to-Peer Networking and Applications*, 10(5):1232–1256, 2017.

[212] Jianying Zhou, Robert Deng, and Feng Bao. Some remarks on a fair exchange protocol. In *International Workshop on Public Key Cryptography*, pages 46–57. Springer, 2000.

[213] Yousaf Bin Zikria, Sung Won Kim, Oliver Hahm, Muhammad Khalil Afzal, and Mohammed Y. Aalsalem. Internet of things (iot) operating systems management: Opportunities, challenges, and solution. *Sensors*, 19(8), 2019. ISSN 1424-8220. doi: 10.3390/s19081793. URL https://www.mdpi.com/1424-8220/19/8/1793.