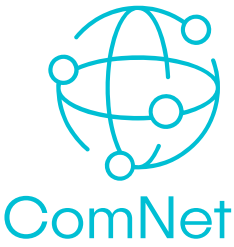


ComNets

TUHH
Institute of
Communication
Networks

MASTER'S THESIS
MAHDI SAFAHIEH

—

Evaluation of the Impact of 6P Transaction Failures on 6TiSCH Wireless Sensor Networks

November 28, 2023

tuhh.de

Evaluation of the Impact of 6P Transaction Failures on 6TiSCH Wireless Sensor Networks
Bewertung der Auswirkungen der 6P-Transaktion Ausfälle in drahtlosen 6TiSCH-Sensornetzwerken

Mahdi Safahieh
Matriculation number: 11861335
Computer Science M.Sc.

Hamburg University of Technology
Institute of Communication Networks
First examiner: Prof. Dr.-Ing. Timm-Giel
Second examiner: Prof. Dr. Marko Lindner
Supervisor: Yevhenii Shudrenko

Hamburg, November 28, 2023

Declaration of Originality

I hereby declare that the work in this thesis was composed and originated by myself and has not been submitted for another degree or diploma at any university or other institute of tertiary education.

I certify that all information sources and literature used are indicated in the text and a list of references is given in the bibliography.

Hamburg, November 28, 2023

Mahdi Safahieh

Abstract

Congestion-free communication and energy-efficient operation are essential requirements in wireless sensor networks. Achieving these goals necessitates the development of communication standards and mechanisms that can effectively manage network resources. The IEEE 802.15.4e standard introduces Time-Slotted Channel Hopping (TSCH) as a solution to address these challenges. TSCH, which is a fundamental component of the 6TiSCH protocol, relies on a negotiation process facilitated by 6P transactions to establish and manage communication schedules, ensuring efficient resource utilization.

However, like various other communication protocols, 6P transactions are susceptible to failures, which can significantly impact network performance. In this thesis, an investigation was conducted to understand the consequences and impacts of 6P transaction failures on 6TiSCH network performance and to determine how parameters like the timeout value and schedule length influence these outcomes. Additionally, the analysis was extended to evaluate the impact of schedule inconsistency on schedule adaptation time and, consequently, on network performance. Key Performance Indicators (KPIs) were utilized in this study to measure and analyze the impact of these factors. In this study, randomly generated 6P response failures within the range of loss probabilities were deliberately induced to comprehensively evaluate their effects on network performance. The experiments were conducted using the Contiki-NG operating system on the OpenMote-B platform, allowing accurate emulation of various scenarios and the gathering of empirical data for analysis. The analysis revealed strong correlations between 6P transaction failures, schedule inconsistency, and the degradation of key performance indicators. This leads to the conclusion that unsuccessful 6P transactions can significantly impact the performance of the 6TiSCH network.

Contents

1. Introduction

Wireless Sensor Networks (WSN) [1] has been recognized as a revolutionary technology within the domain of the Internet of Things (IoT) and industrial applications. These networks play an important role in collecting data from various sensors and transmitting it wirelessly, enabling real-time monitoring and control of physical processes [2]. The effectiveness and performance of such networks are critical factors in determining their success. One of the key standards that has significantly contributed to the advancement of WSNs is IEEE 802.15.4 [3]. This standard provides a foundation for device communication by defining the physical and media access control layers for Low-Rate Wireless Personal Area Networks (LR-WPANs). To enhance the reliability and efficiency of data transmission within WSNs, TSCH is often employed. TSCH synchronizes the operation of sensor nodes by dividing time into slots and assigning specific channels for communication, thus reducing interference and packet collisions. The utilization of the IEEE 802.15.4 standard in conjunction with the Time-Slotted Channel Hopping protocol presents a number of significant benefits, including better reliability, optimal energy consumption, and predictable communication schedules. This guarantees that WSNs are able to function optimally in a wide range of demanding and varied situations, including industrial settings, smart cities, and environmental monitoring systems [4]. 6TiSCH [5], which stands for IPv6 over Time-Slotted Channel Hopping mode of IEEE 802.15.4e, enhances the capabilities of Wireless Sensor Networks . This protocol stack is based on the IEEE 802.15.4 standard and employs the 6top sublayer to optimize resource allocation, ensure timely data delivery, and manage communication schedules among devices within the network. With 6TiSCH, WSNs can operate more efficiently and effectively in complex and dynamic environments.

Wireless Sensor Networks play a significant role in real-time applications such as healthcare monitoring [6] and industrial processes. In such applications, it is crucial to maintain uninterrupted data flow for reliable data transmission. Communication protocols and network sublayers, such as 6TiSCH, are critical components that must function as expected to ensure the reliability of data transmission. Any disruptions or failures in these components can lead to severe consequences such as data loss, communication delays, or even system-wide malfunctions. Therefore, it is vital to have a comprehensive understanding of potential failure scenarios and evaluate their impact on the network's overall performance. This understanding will help develop and implement effective strategies to mitigate such failures and facilitate rapid recovery, ensuring the network's resilience and reliability.

1.1. Problem statement

The 6top sublayer is a crucial component of the 6TiSCH protocol stack that plays an essential role in ensuring the reliability of Wireless Sensor Networks. Its primary responsibility is to allocate resources, through the use of 6P transactions. These transactions allow nodes to coordinate and agree on the addition or removal of communication resource units. However, similar to other network protocols, 6P transactions are susceptible to failures resulting from various factors, including communication errors, node malfunctions, and interference. Such failures can have significant consequences.

Understanding the effects of 6P transaction failures is crucial for ensuring the resilience and dependability of 6TiSCH networks. In this thesis, the objective is to conduct experiments and evaluate KPIs to identify the multifaceted consequences that can result from these failures. By comprehensively assessing these implications, insights can be gained into how these failures affect the overall network performance.

1.2. Thesis Outline

The remainder of this thesis is organized as follows: In Chapter 2, the conceptual foundations of 6TiSCH will be explored, and the complexities associated with 6P transactions will be delved into. This chapter will address potential issues that may arise from 6P transactions and provide detailed information about the 6TiSCH protocol stack. Following this, it is essential to explore the related work in the field. In Chapter 3, an examination of prior research, studies, and findings relevant to the investigation will be conducted. Chapter 4 outlines the research methodology, providing a general view of the experimental setup, data collection methods, the software and hardware tools used, and the key performance indicators that will be employed to evaluate the research findings. In Chapter 5, the implementation details will be explored, covering the testbed setup, simulation of 6P transaction failures, the states of transactions, and the challenges faced during the implementation, particularly within the sf-simple framework. This chapter provides valuable insights into the practical aspects of the research. In Chapter 6, the results will be presented and analyzed meticulously, providing valuable insights and interpretations. Following this, Chapter 7 will be dedicated to the discussion of the conclusion.

2. Background

In this chapter, we explore deeper into the fundamental ideas and developments that serve as the foundation for this master's thesis. The background information that is presented in this section lays the groundwork for a more in-depth comprehension of the chapters that are to follow. We begin by providing an overview of Wireless Sensor Networks, which serve as the primary foundation for our research. As a continuation of this discussion, we will dive into details of the 802.15.4 Time Slotted Channel Hopping protocol, which is an essential component found within the world of WSNs. In order to expand our knowledge even further, we look into the integration of Internet Protocol version 6 (IPv6) over the TSCH mode of IEEE 802.15.4e, which is more commonly referred to as 6TiSCH. We focus on the most important sublayers and functions, such as the 6TiSCH Operation Sublayer (6top), the 6TiSCH Minimal Scheduling Function (MSF), and 6P Transactions.

2.1. Wireless Sensor Networks

Wireless Sensor Networks are a group of interconnected nodes, known as sensor nodes, that communicate wirelessly with each other. Each sensor node has four essential components: a sensor for collecting data, a microcontroller for processing data, a radio transceiver for wireless communication, and a power supply, typically in the form of a battery. These physical features allow sensor nodes to work together in sensing and collecting information within their designated network coverage areas. These networks are designed for real-time monitoring and data collection of various physical and environmental parameters, such as temperature, humidity, vibration, and motion. The data collected by sensor nodes is sent to a central sink node [7], which is defined as a specialized node responsible for collecting, aggregating, and centralizing the gathered data. This sink node subsequently transmits the data to a central user for further processing, analysis, and potential action initiation. The communication between WSN layers is depicted in Figure 2.1. WSNs are highly valuable because they can function in energy-constrained environments [8], making them essential in a wide range of applications and industries. These applications include logistics management, vehicle routing optimization, military operations support, environmental monitoring, healthcare, and smart home automation, among others [9]–[11].

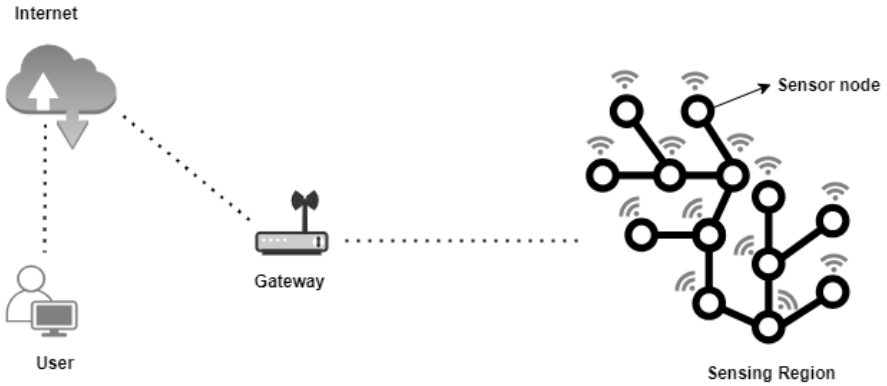


Figure 2.1.: Wireless sensor network.

2.2. IEEE 802.15.4 Standard

Wireless Sensor Networks are well-known for their self-organizing capabilities, scalability, and energy efficiency, which allow them to function effectively in dynamic environments while conserving limited battery power. To ensure the seamless operation of WSNs, it is essential to use specific standards and mechanisms. One key standard at the core of WSN operations is IEEE 802.15.4. This standard was developed by the IEEE and serves as the basis for low-power, low-data-rate, and cost-effective wireless communication within WSNs [12]. IEEE 802.15.4 has been designed to meet the diverse needs of WSN applications. It defines the physical (PHY) and Medium Access Control (MAC) layers of the protocol [13], thereby enabling devices to communicate with each other efficiently. However, it is worth noting that IEEE 802.15.4 has certain limitations that have been identified through research and practical implementations. These limitations include challenges related to communication reliability, unbounded delays, interferences, etc [14]. In response to these identified limitations, the 802.15 Task Group 4e introduced an amendment to the IEEE 802.15.4 standard in 2012. This amendment, known as IEEE 802.15.4e, aimed to address the existing limitations and redesign the 802.15.4 MAC protocol. The IEEE 802.15.4e amendment introduced enhancements, such as time-slotted channel hopping in the MAC layer, to improve the reliability and efficiency of WSNs and make them even more suitable for a wide range of applications.

2.3. Time Slotted Channel Hopping

The Time Slotted Channel Hopping protocol, which is a component of the IEEE 802.15.4e standard (an extension of IEEE 802.15.4), is a channel access method for shared-medium networks. TSCH uses a combination of time-division and frequency-division mechanisms (TDMA/FDMA) to ensure dependable, energy-efficient, and predictable communication

[15]. TSCH divides time into fixed duration units called timeslots. The duration of each timeslot is usually 10ms which is long enough to transmit a packet and receive its acknowledgment [16]. Each node uses timeslots for the purpose of broadcasting, transmission, and receiving data. The synchronization mechanism in timeslots ensures that all nodes in the network have coordinated time slots, allowing them to precisely time their transmissions and receptions. At the beginning of the timeslot, data packets are transmitted exactly after Startup Time (Known as TsTxOffset), while the receiver node starts listening to a GuardTime before to account for slight desynchronization. If the reception doesn't begin within this time, the node saves energy by turning off its radio. This synchronization ensures that nodes are active when they need to be and idle when they don't, allowing for efficient use of energy resources within the network. A group of timeslots creates a slotframe that repeats over time. Each time a slotframe cycle begins, nodes have another opportunity to synchronize their communications within the network, maintaining precision and order. This cyclical repetition ensures efficient data exchange and coordinated operations. In Figure 2.2, the TSCH timeslot structure and an illustrative example of a slotframe are depicted.

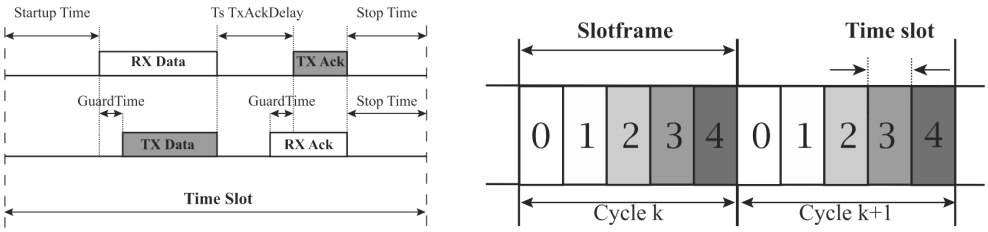


Figure 2.2.: TSCH timeslot and example slotframe [17].

The IEEE 802.15.4e TSCH protocol makes use of the multiple available communication channels to overcome issues like interference and multi-path fading that may affect a particular channel [18]. It enables a node to change its communication channel after every timeslot, a process called channel hopping. The determination of the physical channel for transmission is dependent on the channel offset, which is calculated using the following formula [19] :

$$\text{Physical Channel} = (\text{ASN} + \text{ChannelOffset}) \mod \text{NumCh} \tag{2.1}$$

Where The ASN, or absolute slot number, is the number of timeslots that have passed since the network was set up. The Physical Channel tells the hopping sequence list which channel to use, while NumCh shows the number of channels are available in total. The 802.15.4 PHY framework has 16 non-overlapping channels that can be used in the hopping list [19]. By using this equation, TSCH strengthens communication dependability by using a slow channel hopping strategy to combat noise and interference. For one

slotframe, link-scheduling is utilized to schedule a link, which is a pair of time (slotOffset) and frequency (channelOffset) for communication between two nodes, referred to as a "cell," in the communication schedule. The communication schedule of the network is represented as a 2-dimensional matrix (see figure 2.3). Columns represent timeslots, and rows represent channel offsets. The discrete communication resources represented by the matrix's cells can either be shared or dedicated [20]. Dedicated cells are reserved for unicast transmissions, such as data exchange, whereas shared cells are designated for broadcast frames (such as Enhanced Beacons) in the first timeslot. In this context, the

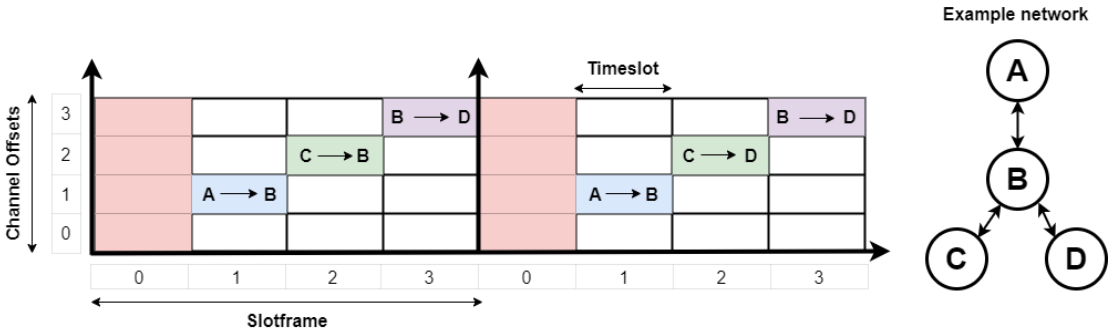


Figure 2.3.: Simple Network topology with example TSCH schedule.

coordinator nodes take on the responsibility of periodically broadcasting TSCH frame call Enhanced Beacons. These Enhanced Beacons are used for advertising the network and contain information like the Absolute Slot Number (ASN) to maintain synchronization and mitigate external interference. This ASN ensures nodes agree on the current timeslot in the slotframe by correlating to all of the cumulative timeslots since the network's inception. TSCH scheduling's flexibility enables the development of specialized TSCH schedulers tailored to various network requirements as well as customized solutions.

2.4. 6TiSCH (IPv6 over IEEE 802.15.4e TSCH)

In 2013, the IETF Working Group 6TiSCH was formed with the aim of allowing the use of IPv6 on the IEEE802.15.4e TSCH link layer [21]. The group's efforts led to the establishment of standards for both the control plane and the adaptation of the IP layer over this link layer. By utilizing IPV6 and its routing capabilities, 6TiSCH Architecture can efficiently scale the network for maximum simplicity and productivity [22]. The 6TiSCH architecture has a network stack (shown in figure 2.4) that enables IPv6 connectivity, built on a solid foundation. The network stack includes important

components in the upper layers that work together to facilitate effective communication. These components are 6LoWPAN, RPL, and CoAP, each with its own specific role.

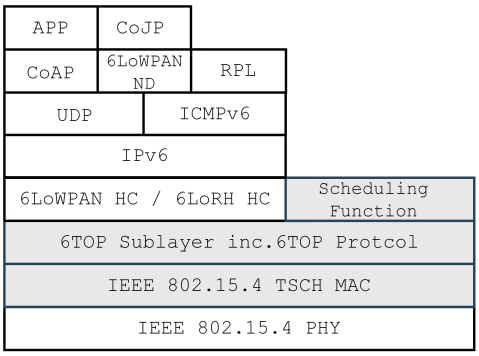


Figure 2.4.: 6TiSCH Protocol Stack.

2.4.1. 6TiSCH Operation Sublayer

The 6TiSCH has been improved with the addition of the 6top sublayer. This sublayer is responsible for managing and optimizing network performance. It operates within the 6TiSCH stack's link layer and coordinates the scheduling of transmissions and receptions between nodes in the network. The term "6top" is short for "6TiSCH Operation Sublayer". The 6top sublayer collaborates with higher-layer protocols like Routing Protocol for Low-Power and Lossy Networks (RPL) and Constrained Application Protocol (CoAP). This integration helps the network to improve routing decisions and application-specific communication requirements in sync with the underlying link layer scheduling. The 6top sublayer within the 6TiSCH architecture is composed of two fundamental components: the Scheduling Function (SF) and the 6P (6top Protocol). These components work together to manage communication slots and facilitate dynamic scheduling within low-power and lossy wireless networks. The following subsections will discuss the details of each of these components.

2.4.2. 6top Protocol (6P)

As mentioned earlier, network orchestration in 6TiSCH is accomplished through scheduling, and the creation of such a schedule requires a mechanism that enables nodes in the network to reach a consensus on communication cells. The 6P Protocol, often referred to simply as 6P, stands as one of the crucial components of the 6top sublayer. Defined in RFC 8480, this protocol introduces an innovative approach to control plane operation by enabling pairwise negotiations between neighboring nodes. Its primary objective is to facilitate

the seamless allocation of cells within the communication schedules of these neighboring nodes. Two neighboring nodes can modify their communication schedule by initiating a 6P negotiation, either adding or deleting a communication cell as needed. All commands supported by 6P can be found in Table 2.1.

Command	Code	Description
ADD	1	Add cell(s) between two neighbors
DELETE	2	Delete cell(s) in the schedule
RELOCATE	3	Relocate cell(s) in the schedule
COUNT	4	Count the cells
LIST	5	List the cells
SIGNAL	6	Placeholder for SF-specific commands
CLEAR	7	Clear all cells between two neighbors

Table 2.1.: Commands supported by 6p.

2.4.3. 6P Transactions

As mentioned earlier, for dynamic scheduling and coordination of network resources, nodes require a series of interactions and communications, known as '6P transactions.' These interactions involve a handshake-like mechanism where they negotiate the arrangement of cells within their TSCH schedule, deciding on additions, deletions, or relocations. In 6P transactions, sequence numbers are used as a mechanism to guarantee schedule consistency between two neighboring nodes. Maintaining this consistency is essential as any deviation may lead to a potential loss of connectivity. We will discuss this mechanism in detail in future sections. The 6P transaction process can be executed in either a 2-step or 3-step manner. In a transaction with 2-steps, the initiating node chooses the cells for allocation,, while in a 3-step transaction, the recipient is given the opportunity to make the cell selections. Figure 2.5 illustrates examples of both 2-step and 3-step 6P Transactions. To clarify the functionality of the 6P transactions, we will focus on explaining the 2-step example.

In the provided example for a 2-step transaction, Node A's scheduling function concludes that, due to traffic load, there is a need to include two additional cells in the schedule. Node A suggests three cells for B to choose from. These suggested cells are locked by Node A once the 6p response sent by Node B is received. Node A initiates a 6p Add request to Node B by configuring Numcells to 2 and indicating the cell list with 3 potential cells. Receiving a link layer acknowledgment confirms that the request was successfully

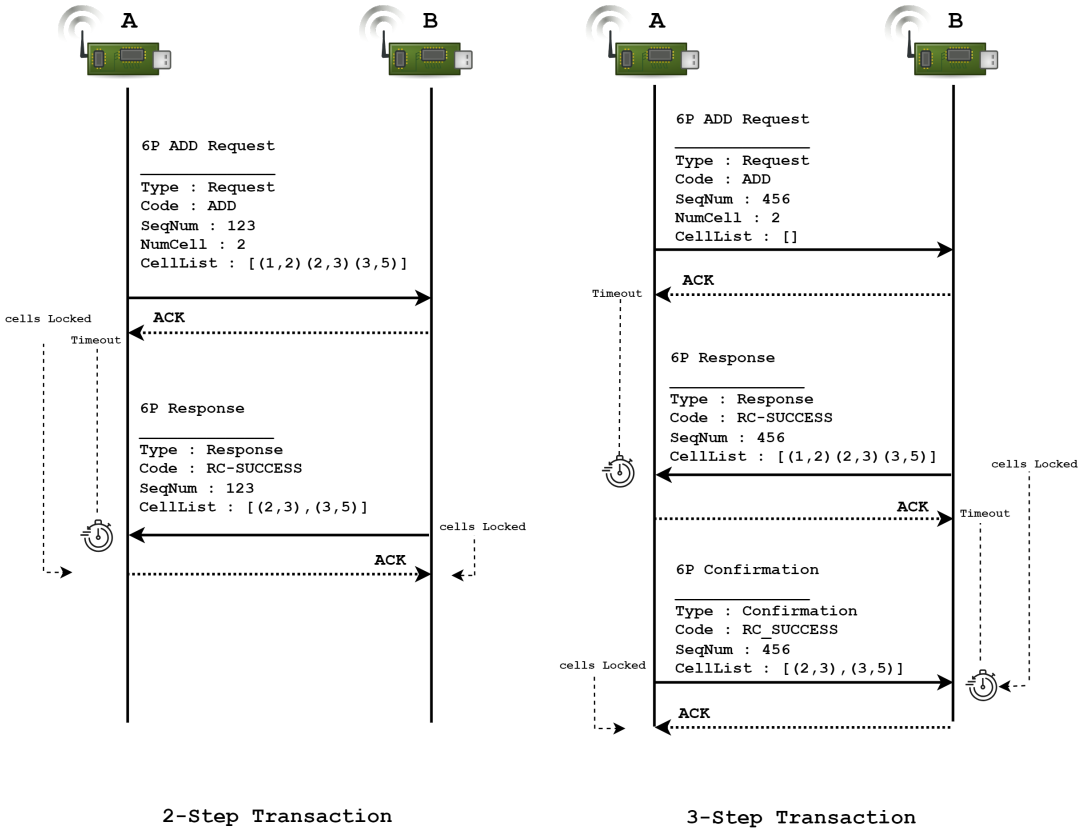


Figure 2.5.: Example of 2-Step and 3-Step 6p Transaction.

received by Node B. Upon receiving the acknowledgment, Node A activates a timeout mechanism, which aborts the 6P transaction in the absence of a response. From the receiver's perspective, the scheduling function on Node B receives the Add request and selects two cells from the cell list provided by Node A. The cells chosen by Node B are locked until a successful response is received by Node A, which is confirmed through an acknowledgment, indicating a successful transmission.

Subsequently, a 6P Response message is generated and sent back to Node A by Node B. If Node A receives the 6P Response message successfully, it acknowledges the reception by sending a link-layer acknowledgment to Node B, ensuring both nodes are in sync regarding the progress of the transaction. Upon the successful completion of this negotiation, Node A and Node B reach an agreement to incorporate the two selected cells into their Time-Slotted Channel Hopping schedules.

2.4.4. 6TiSCH Minimal Scheduling Function

So far, we have seen how the 6P protocol uses 6P transactions to shape each node's schedule by adding or removing dedicated cells. Now, we will discuss the functionality of the scheduling function, which makes decisions about when cells should be added or deleted based on traffic load. For simplicity in further explanations, we will use the abbreviation "SF" for the scheduling function and "MSF" for the minimal scheduling function.

Based on each application's requirements, different SFs might be used. The MSF, known as the default SF, extends the minimal schedule setting. MSF defines bootstrapping policies for nodes joining the network, adapting to traffic changes, and handling schedule inconsistency. MSF uses three types of cells for its oprations: *Minimal cell* , a shared cell that provides minimal connectivity between nodes in the network. minimal cell is used by MSF to broadcast advertising frames like Enhanced Beacons (EB) and routing information in Slotfram 0. *Autonomous cells* are cells initialized by MSF for bootstrap unicast communication in Slotframe 1. These forms of cells are referred to as autonomous because each node generates and maintains them without the need for 6P negotiation. There exist two separate types of autonomous cells , Autonomous Tx Cell (AutoTxCell) and Autonomous Rx Cell (AutoRxCell) [22]. These cells are used to send and receive the initial messages for exchanging key information. *Negotiated cells* are generated through 6P transactions and are utilized for communication between neighboring nodes.

Due to the scope and emphasis of this thesis on "6P transaction failures," we have intentionally omitted in-depth discussions of MSF rules for the bootstrapping join process. It's important to note that other aspects of the MSF are briefly addressed within this research context, especially those directly related to or contributing to the understanding of 6P transaction failures, such as "Rules for Adding and Deleting Cells" and "Schedule Inconsistency Handling".

2.4.5. Rules for Schedule Adaptation in MSF

After a Node Becomes part of the 6TiSCH network, MSF dynamically adjusts the communication schedule for each node. This schedule adaptation is carried out for three specific purposes: Adapting link-layer resources to the traffic load between a Node And its neighbor, new parent selection, and occurrence of a schedule collision

1. **Cell Adaptation to Traffic changes**

   The MSF operates with the aim of efficiently managing the communication schedule within the 6TiSCH framework, employing a distributed approach. Each node in the network actively optimizes its allocated cells by monitoring their utilization in interactions with its neighbor. To adapt to varying network conditions, a node

dynamically adjusts the number of negotiated cells. This adjustment occurs when the node detects a substantial increase or decrease in network traffic. When there's an increase in traffic, the node decides to add a new cell to accommodate the higher rate of exchanged link-layer frames with its neighbor. Conversely, if the traffic decreases, the node decides to remove a cell, ensuring that resources are efficiently allocated even as the communication demands vary within the 6TiSCH schedule. This schedule adaptation is done by using two counters (*NumCellsElapsed , NumCellsUsed*) [22]. *NumCellsElapsed* counts how many negotiated cells have passed since the counter's initialization, while *NumCellsUsed* determines how many negotiated cells have been used for transmission. When the value of *NumCellsElapsed* reaches *MAX_ NUM_ CELLS*, a decision-making point is reached. At this point MSF makes the decision to Add or Delete a cell by measuring the following ratio:

$$Cell\text{-}Usage = \frac{NumCellsUsed}{NumCellsElapsed}$$

MSF sends a 6P Add request for dedicating a cell if *Cell-Usage* is greater than the High threshold (75%) and a 6P Delete request for cell usage less than the low threshold (25%). Default values for Thresholds are recommended in [23].

2. **Switching Parent**

When a node decides to change its parent due to varying link quality [24], MSF on the node should be able to take action to switch from the old parent to the new parent. Node Knowing the number of negotiated cells in its slotframe triggers one or many 6P Add requests to establish a new schedule with the same number of dedicated cells for its new parent. When the procedure of creating the new schedule is successful, a 6P CLEAR request must be sent to the old parent to remove the previously negotiated cells.

3. **Handling Schedule Collisions**

Schedule collision occurs when neighboring nodes schedule a negotiated cell in the same timeslot location, specified by [slotOffset, channelOffset], within the TSCH schedule. Such overlap can result in data exchange collisions between nodes. To address this issue, MSF needs to possess the capability to manage these collisions effectively. One crucial aspect of managing schedule collisions is the tracking of transmission attempts and successful acknowledgments. This tracking is essential to understand the reliability of data transmissions. To facilitate this tracking, each node must maintain two counters, namely *NumTx* and *NumTxAck*, for every negotiated TX cell [25]. Here, *NumTx* counts the number of transmission attempts on the cell, while *NumTxAck* represents the count of successful transmissions on the cell with acknowledgments. During each iteration, both counters are incremented by one.

The ratio $\frac{NumTxAck}{NumTx}$ serves as the packet delivery ratio (PDR). By calculating this ratio for each cell, a node can identify instances of collision. The default PDR threshold, denoted as *RELOCATE_PDRTHRES*, is preset to 50%. Cells exceeding this threshold are deemed to have a high collision rate. In such scenarios, the node initiates the relocation process by invoking the 6P RELOCATE command.

2.5. 6P Timeout

"6P Timeout" refers to the amount of time a node in a TSCH network will wait for a response after sending a 6P request. If a timeout occurs, the node may either retry the request or take appropriate action based on the network's policies. Finding an optimal 6P timeout value is crucial in the 6TiSCH network and must be chosen carefully. An increased timeout value can potentially enhance transaction rates. However simultaneously leads to the retransmission of already received messages, causing a Sequence number inconsistency. On the other hand, A shorter timeout value, while decreasing transaction delays and total exchanged messages, can unexpectedly increase these inconsistencies [26]. Therefore, balancing the optimal success rate, minimizing network congestion, and minimizing network inconsistencies due to retransmissions and desynchronization necessitate an optimal timeout value. This optimal setting is primarily contingent on the network's architecture, traffic load, and specific system requirements.

2.6. Sequence number Management

The Sequence number in the 6Top sublayer is essential for preserving 6P transaction consistency. It keeps track of the exchange of data order between nodes. Every node in the network maintains a record of the Sequence number (SeqNum) that has recently been utilized by each of its neighbors. This SeqNum is essential for identifying and resolving conflicts in schedules and duplicate commands. An error is reported if a difference in the SeqNum is found, possibly as a result of a lost response message. This mechanism helps to maintain the integrity and dependability of data transfer within the network by enabling the system to identify and respond to message loss or corruption.

During a 6P Transaction, if there is a mismatch between the sequence numbers used by the initiator (Node A) and the receiver (Node B), Node B will send an error message labeled *RC_ERR_SEQNUM* back to Node A. This message is to notify Node A that there is an inconsistency in the sequence numbers, and the transaction cannot proceed until it is fixed. After receiving the error message, Node A's SF decides the best way to handle the issue. This may involve retrying the transaction with the correct sequence number or taking other steps to synchronize the sequence numbers between the nodes. In continuation of this section, we will address two well-known issues related to Sequence Number Management.

2.6.1. Detecting and Handling Duplicate 6P Messages

This issue refers to a condition where a node receives a double 6P Request, Response, or Confirmation. This condition can happen due to a loss of link layer acknowledgment. As depicted in Figure 2.6, after Node A sends a 6P Request with a SeqNum equal to 100 and receives its acknowledgment, Node B sends back its response with the same SeqNum (100). However, during this ongoing transaction, an issue arises when the Acknowledgement from Node A is lost due to poor link quality. At this point, Node B assumes that its response wasn't received by Node A and retransmits the response message with the previous SeqNum. At this time, Node A detects a duplicate 6P message with the same SeqNum and the same message type from Node B. According to [RFC 8480], such an issue must be handled in a way that a node detecting a 6P duplicate Response must send an acknowledgment but has to silently ignore the 6P message.

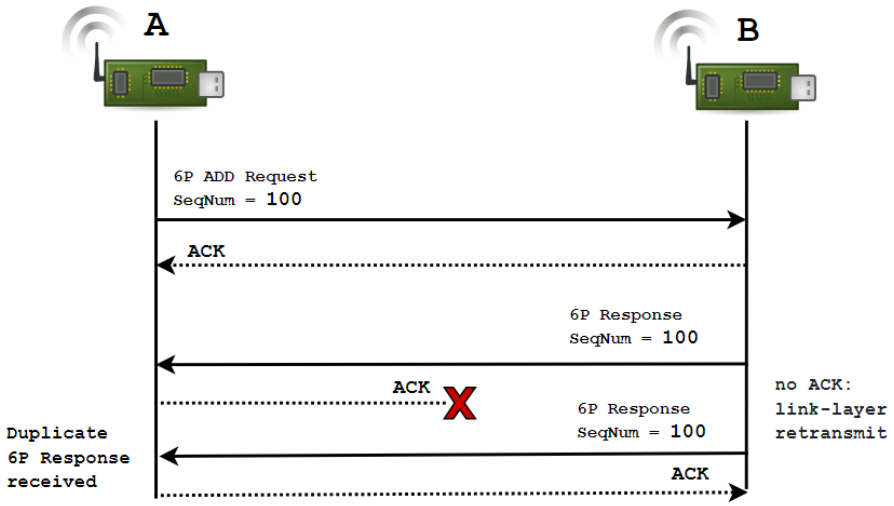


Figure 2.6.: Example of Detecting Duplicate 6P Messages.

2.6.2. Detecting and Handling a Schedule Inconsistency

When the planned communication schedules of two nodes, Node A and Node B, don't match or synchronize correctly, it results in a schedule inconsistency. In other words, Node A and Node B have conflicting schedules. For example, Node A has a scheduled TX cell for transmitting data to Node B. However, Node B's schedule doesn't include a corresponding RX cell for receiving data from Node A. This means Node A is ready to send information to Node B, but Node B isn't set up to receive it at the same time. As a result, their schedules don't align, leading to schedule inconsistency.

The consequence of this inconsistency is a loss of connectivity between Node A and Node B. Since Node B isn't listening in the expected timeslot, the data sent by Node A goes unheard, causing communication failure between the two nodes.

To detect any potential inconsistencies, a node must compute the expected SeqNum value for the next 6P transaction. If a 6P Request is received with a SeqNum value that doesn't match the expected value, it signifies the detection of a scheduling inconsistency. The increment-by-one rule guarantees that messages are processed in a specific sequence, making it easy to identify any inconsistencies. There are two cases in which a schedule inconsistency occurs :

A. Node Loses its state (ex. Power Cycle)

As illustrated in Figure 2.7, Sometimes, a node may lose its operational state when it's reset, like during a power cycle. This can cause the node's SeqNum value to reset to 0. Node A, which has no information about the state of Node B, will increment its SeqNum and send its next request to Node B. In this scenario, Node B will detect the difference in SeqNums and respond with an error response, indicated by *RC_ERR_SEQNUM*.

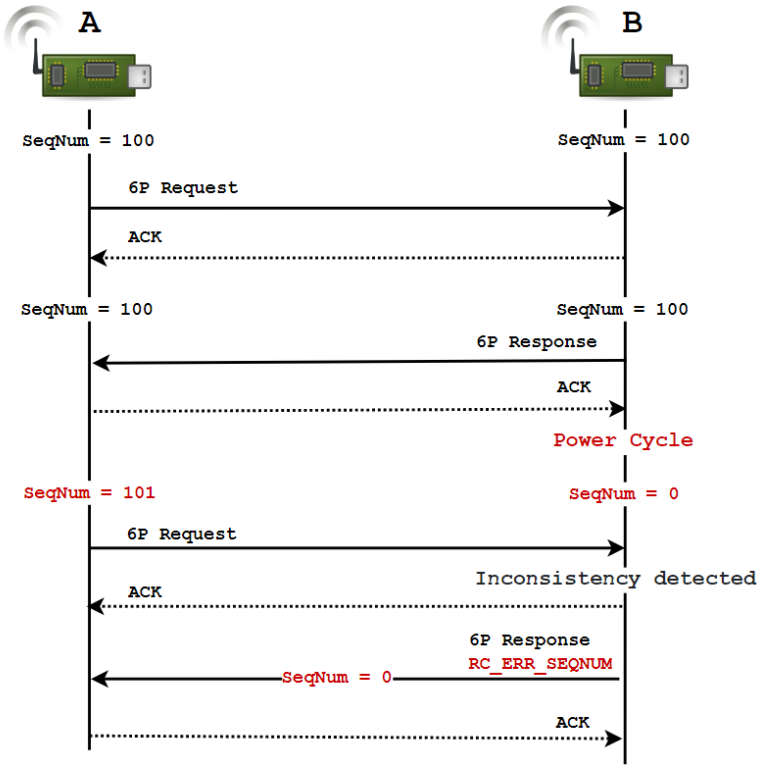


Figure 2.7.: Schedule Inconsistency due to Power Cycle.

B. Hit Maximum Ack-Retransmissions

In the second scenario, Node B will detect an inconsistency if it does not receive an acknowledgment for its initial Response message. It will make multiple attempts to resend the Response, up to a predetermined maximum number of retransmissions. If the maximum retransmission limit is reached without receiving an acknowledgment, it can be concluded that an inconsistency has indeed occurred. Figure 2.8 depicts the procedure for Maximum 2 retransmissions. In all inconsistency scenarios, Node B must send a Response in a 2-Step transaction and a confirmation in a 3-Step Transaction, both with the error code *RC_ERR_SEQNUM*. The behavior for dealing with such inconsistencies is defined by the SF of the node that detects the inconsistency. According to [RFC 8480], SF can handle inconsistencies in three ways:

- Clear the schedule and rebuild it by sending a 6P CLEAR Request.
- Recover the schedule by sending a LIST Request.
- Internally 'Roll Back' the schedule.

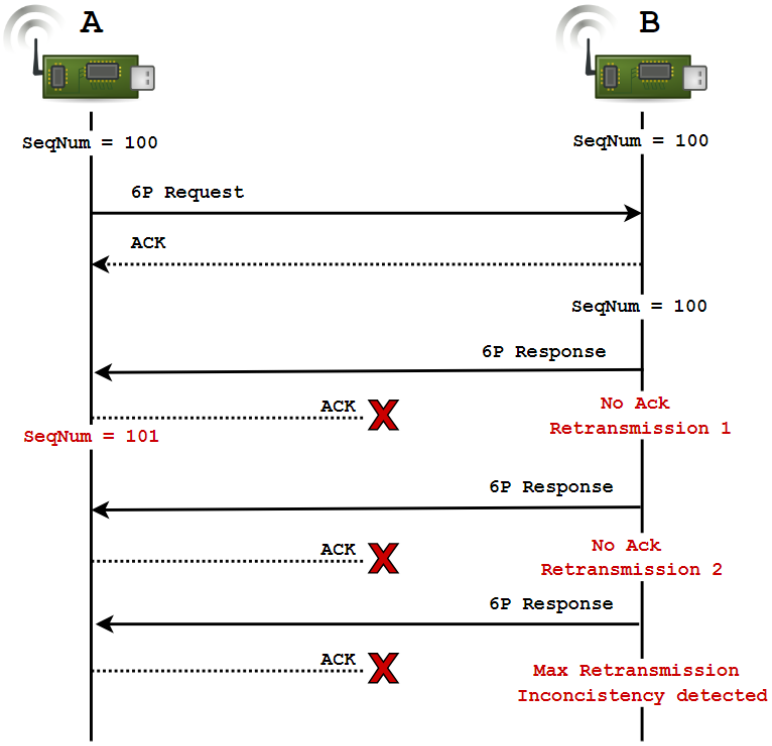


Figure 2.8.: Inconsistency due to Max-Retransmission.

3. Related Work

In this chapter, we will review and analyze existing literature and research related to our topic. By examining prior studies and investigations, we can better understand the context and background of our current study. Our aim is to identify gaps, contradictions, and opportunities for advancing our understanding of the topic.

3.1. Performance Analysis and Optimizations in 6TiSCH

The performance of 6TiSCH networks has been a topic of great interest in recent years. In this section, we will delve into the research that has been conducted on performance metrics, protocols, and optimization techniques in relation to these networks. By studying this area, we can gain valuable insights into the challenges faced by researchers and the solutions that have been proposed, which can serve as a foundation for our own research efforts.

The paper [27] addresses a critical challenge in the 6TiSCH standard protocol: the high network formation time and associated energy consumption. These issues not only hinder network efficiency but also significantly drain the battery life of network nodes. The primary aim of this research is to reduce network formation time within the 6TiSCH framework, focusing on TSCH synchronization time and energy consumption. To tackle this challenge, the authors introduce a novel scheduling function, SF-Fastboot, designed to streamline communication schedules within the network. This aims to accelerate synchronization and network formation while minimizing energy consumption. A core principle of SF-Fastboot is exploiting the asymmetry between the root node and other nodes. SF-Fastboot allows the root to send beacons more frequently than other nodes without significantly affecting energy consumption or network lifespan. It introduces a specific channel offset (ch_off_{fastboot}) for network formation and continuously schedules beacon TX cells on multiple channels at this offset, reducing the time needed for joining nodes to synchronize with the network. Once synchronized, joining nodes initiate the CoJP joining process by transmitting join requests to the root, and the root schedules RX cells to distribute these requests evenly and prevent congestion. SF-Fastboot effectively manages beacon transmissions and minimizes keep-alive traffic, leading to more efficient network operation. To evaluate the effectiveness of SF-Fastboot, simulation tests were conducted. The performance measures included TSCH synchronization time, network formation time, and energy consumption. The results were compared with the performance of vanilla MSF under the same parameters. The results demonstrated that SF-Fastboot

offers significant improvements in network formation time by 41% – 80% compared to MSF. In terms of energy consumption, SF-Fastboot reduced energy consumption by 30% – 32% in fully-meshed placement and by 29% – 38% in random placement. The paper also discusses how SF-Fastboot can be integrated with MSF with minor adjustments. Evaluation results demonstrate that SF-Fastboot significantly improves network formation time, reduces TSCH synchronization time, and lowers energy consumption when compared to MSF. In conclusion, SF-Fastboot presents a promising solution to the challenge of lengthy network formation time within the 6TiSCH framework, significantly improving network efficiency and energy consumption.

Authors of [28] focus their research on analyzing the performance of the On-The-Fly (OTF) Bandwidth Reservation Algorithm within the 6TiSCH architecture[29], as the reference SF. The aim of their study was to investigate how OTF's performance is impacted by the RPL and 6top negotiations through a research methodology that involves simulation using the Contiki OS and Cooja simulator. The study reveals that as network traffic increases, the OTF algorithm's reliability decreases, and latency rises significantly, making it unsuitable for critical industrial applications. Adjusting a key parameter, the hysteresis quantum (T), provides limited improvements. Queue dynamics show that initial timeslot negotiations cause queue growth and changes in preferred parent nodes lead to persistent queue issues. RPL protocol dynamics and link-quality assessment introduce frequent parent changes and poor link choices, worsening queue problems. Additionally, disruptive events like 6top transaction failures and TSCH de-synchronization lead to schedule resets, node disconnections, and increased delays, further impacting network performance. In response to these challenges, the authors propose a modified version of OTF, named Enhanced-OTF (E-OTF), which incorporates a mechanism for rapid congestion recovery. E-OTF modifies the Allocation Algorithm to consider channel quality and react to queue changes. E-OTF solution effectively addresses congestion, reducing latency and packet loss. It proves highly effective in handling parent changes and 6top failures with minimal additional energy consumption.

3.2. Innovative MSF improvements

The evaluation and improvement of MSF performance has been the subject of numerous research projects, all of which have helped us understand its strengths and weaknesses. In this section, we will delve into the extensive body of research that has examined the functionality of MSF and introduce innovative improvements intended to boost its performance. Through this comprehensive review, we hope to present a gathered overview of the developments made in this important area and throw light on the creative approaches developed to utilize the Minimum Scheduling Function to its fullest potential in contemporary wireless networks.

In the research conducted by Chang et al. in 2020 [30], the study primarily centers on the optimization of MAX_NUMCELL within the MSF to reduce 6P message overhead

in 6TiSCH wireless sensor networks. It seeks to address the critical challenge of setting MAX_NUMCELL effectively to improve network efficiency and minimize transaction overhead while maintaining low end-to-end latency. The core issue the research tackles is the absence of clear guidelines for configuring the MAX_NUMCELL parameter in the 6TiSCH protocol stack. MAX_NUMCELL plays a pivotal role in determining resource allocation, specifically the number of cells in the TSCH schedule, and its misconfiguration can lead to unnecessary 6P message generation. The key results and achievements of this study are the recommendations for setting MAX_NUMCELL values based on traffic patterns. For periodic traffic, the research suggests that configuring MAX_NUMCELL at least four times the traffic load can notably reduce the number of 6P messages, improving network efficiency. On the other hand, for bursty traffic scenarios, lower MAX_NUMCELL values are found to minimize end-to-end latency, though this comes with the trade-off of generating a higher number of 6P messages. Additionally, the research introduces an enhanced protocol, A-MSF, designed to streamline the process of cell addition and deletion. A-MSF proves to be effective in significantly reducing the number of 6P messages, achieving a remarkable 44% reduction in communication overhead compared to MSF, while still maintaining low end-to-end latency.

The paper [31] by Karnish et al. provides an investigation into existing inefficiencies of the Minimal Scheduling Function. MSF was identified as having several limitations including a tendency to be reactive rather than proactive in managing traffic changes and notably, a conservative adaptation strategy where adjustments are made by a single cell even when facing significant changes in traffic requirements. These shortcomings resulted in a high signaling overhead and limited end-to-end latencies. This research, hence, was driven by the goal of mitigating these issues with the introduction of an Improved Minimal Scheduling Function designed to optimize Link Scheduling in 6TiSCH Networks. The primary objectives of the proposed IMSF were to dynamically meet traffic requirements, significantly decrease the control overhead associated with 6P signaling, and improve end-to-end latency and reliability. With IMSF, authors addressed the limitations of MSF by estimating the number of cells required for communication in the forthcoming cycle and then using this estimation to dynamically adjust the network to correspond with this projected traffic. Furthermore, IMSF uses thresholds to determine whether to add or remove cells from its schedule, allowing for faster adaptations in response to changing traffic demands. The IMSF scheme also accounts for certain essential factors such as the Expected Transmission Count and node queue occupancy, which allows for more accurate cell computation. The efficacy of the developed IMSF was evaluated employing a comparative approach, where its performance was tested against the existing A-MSF and MSF schemes using a 6TiSCH simulator. For a comprehensive evaluation, several critical factors were considered: changing network sizes, varied traffic rates, and diversified network topologies. Through the simulated experiments, the efficiency of IMSF was clearly demonstrated. IMSF consistently outperformed both A-MSF and MSF across all the tested scenarios. In the instance of periodic traffic where each node generated 2 packets per slotframe, IMSF successfully minimized 6P control overhead, reduced end-to-end latency and effectively managed packet loss. IMSF also exhibited adaptability in diverse

network topologies including linear, random, and tree, reducing 6P overhead significantly even in high-traffic scenarios. Notably, as traffic rate increased, IMSF effectively reduced the 6P transactions overhead by 25% and 48% compared to A-MSF and MSF.

In the study presented in [32], the authors aimed to evaluate the performance of MSF to understand its behavior regarding resource allocation in response to changes in traffic levels, ranging from low to high. MSF possesses two critical abilities essential for ensuring network reliability: the capacity to adapt its scheduling based on traffic changes and the capability to reallocate resources in the event of collisions. For their evaluation, the authors considered two traffic patterns: regular and constant, along with a scenario involving varying traffic to assess MSF's adaptation functionality. They utilized a simulation setup, by using 6TiSCH simulator [33]. In the case of constant traffic, they relied on a linear topology in which each node generated regular traffic within a different range. In this scenario, their focus was on a node that was particularly susceptible to suffering from collisions. The results revealed that a higher sending rate would decrease the Packet Delivery Ratio (PDR) due to a full queue in the absence of the required number of cells. Latency also decreased at higher traffic levels because more cells were scheduled, increasing the chances of successful transmission. In another experiment, they assessed MSF's traffic adaptation for both low and high traffic rates. Based on this experiment, they demonstrated that the majority of packet losses occur during cell allocation due to the limited queue size. As this period of cell allocation grows, more packet loss happens. They also showed that MSF scheduled more cells than needed, which is consistent with expectations due to overprovisioning. For changing traffic, they aimed to demonstrate how quickly MSF can adapt the schedule when traffic patterns change. Their results revealed that by passing more slotframes and allocating more cells, the time taken to adapt decreases. In conclusion, the study highlights the effectiveness of the Minimal Scheduling Function in dynamically adapting to changes in traffic levels. It emphasizes the importance of proper resource allocation to maintain network reliability, especially during periods of high traffic.

Kim et al. [34] introduce a novel approach called Delayed Cell Relocation (DCR) to address the issue of schedule collision in wireless sensor networks using the Minimum Scheduling Function. The MSF is a distributed scheduling mechanism that relies on random selection of candidate cells for allocation, and it faces challenges when adjacent nodes allocate cells with the same [slotOffset, channelOffset], leading to interference. This interference can result in schedule collisions, which are typically resolved using the 6P RELOCATE command in the MSF, but this process is not optimal and introduces overhead. The DCR method proposed in the paper aims to enhance the existing MSF mechanism by reducing the relocation overhead. DCR operates by inducing only one pair of nodes, among those experiencing a schedule collision, to relocate their cells. It accomplishes this by introducing delay times based on cell allocation time. As a result, cells that have been in operation for an extended period are less likely to be relocated, which contributes to network stability. The delayed cell relocation scheme calculates the delay time by considering the difference between the Absolute Slot Number assigned

to the cell and the current ASN. This delay time is measured in slotframes, with a maximum value of 256. By making this delay proportional to the allocation time, the paper presents a strategy to reduce the number of relocations, ultimately improving network performance. The paper also introduces the concept of Keep-Alive (KA) packet transmission as a method for detecting whether schedule collisions have been resolved during the delay time. If the PDR of the KA packets increases, it is inferred that another node pair has resolved the collision, and the relocation of the delayed cell can be halted. To evaluate the DCR's performance, the paper presents simulation results. The experiments demonstrate that DCR effectively reduces the number of cell relocations by approximately 25% when compared to the MSF. The results suggest that the DCR approach is particularly advantageous as the network scales and converges, leading to a decrease in the occurrence of schedule collisions. Additional findings from the simulations include insights into how DCR reduces cell relocations and keeps older, more stable cells in operation while focusing relocation efforts on newly allocated cells. Furthermore, the study indicates that the effectiveness of DCR is influenced by the maximum delay time, with larger values contributing to even fewer cell relocations. In conclusion, In conclusion.

3.3. Performance evaluation of 6top protcol

Understanding the factors that can lead to 6top transaction failures is crucial, as these factors can significantly impact network performance. Recognizing these causes is essential for formulating effective strategies and guidelines to mitigate failures and optimize the 6top protocol's performance. To address these concerns, researchers in the study [35], investigated the performance of the 6top protocol in a real-world environment, revealing that the time required for 6top transactions is not negligible, contrary to previous assumptions. To conduct 6top protocol evaluation, the researchers implemented the 6top protocol within the Contiki OS, utilizing two distinct scheduling functions, namely On-The-Fly (OTF) and the PID algorithm [36]. These SFs, with their differing approaches to timeslot allocation, were instrumental in assessing the protocol's performance. Throughout the study, the authors employ a set of key evaluation metrics to gauge the 6top protocol's performance. These metrics include the transaction Success Rate, Transaction Delay, and transaction Failure Rate. However, the Failure Rate is of particular interest, prompting a detailed exploration into the underlying causes of transaction failures. The research findings reveal that transaction delays are influenced by traffic rates, with PID generating significantly more transactions and longer delays. In comparison, OTF, with its more conservative approach, results in lower transaction delays. The study also identifies a considerable difference in transaction volume between OTF and PID, highlighting that PID generates a much higher number of transactions. Crucially, the authors draw attention to the causes of transaction failures. They identify common reasons for these failures, including insufficient resources, timeout expirations, schedule mismatches, and buffer overflows at the MAC layer. Additionally, the study underscores the importance

of considering memory constraints as a factor contributing to some transaction failures. Lastly, transmission errors and collisions are acknowledged as nearly unavoidable sources of transaction failures.

The research carried out by Yuvin Ha and Sang-Hwa Chung [37] recognized that the performance of 6P transactions, which are a key component of 6TiSCH wireless networks, significantly degraded under harsh network conditions. Their examination led them to conclude that heavy traffic loads and frequent packet loss affected 6P transactions detrimentally. These harsh conditions can be prevalent in Industrial IoT services, where heavy traffic and packet loss scenarios often occur. The main issue they identified was that 6P transactions, in their existing form, suffered from a lack of coordination between nodes. Specifically, child nodes generated 6P requests based on their own cell requirements, contributing to network congestion and increasing the chances of packet loss. Additionally, transaction errors occurred frequently, affecting the nodes' ability to maintain a consistent schedule for data exchange, further compromising network stability and overall performance. To resolve these issues, they introduced two key methods:

1. Parent-Initiated 6P Transaction: The aim of this method was to control and manage the traffic load more effectively. By shifting the responsibility of initiating the 6P transaction to the parent node, the issue of indiscriminate 6P requests was mitigated. As a result, traffic could be managed efficiently based on precise cell requirements, which improved network stability under heavy traffic load.

2. Transaction Revert: The purpose of this method was to increase network stability by enabling local recovery (rollback of the schedule) from a failed or mismatched 6P transaction. It was designed to identify and correct any inconsistencies between the sequence numbers of neighboring nodes, preventing abrupt packet loss and enhancing the reliability of the network. experimental results show promising improvements in network performance with the use of these proposed methods. For instance, the Parent-Initiated 6P Transaction was observed to give a 6P transaction success rate that was more than 100% higher, and a data packet Packet Delivery Ratio improvement of up to 22% compared to conventional transaction methods, even in challenging network environments. The authors also tested the performance of these methods in environments with antennas that provide more stable link quality. The results suggested that Parent-Initiated 6P transaction could still lead to improved performance in such stable environments. Furthermore, the improved transaction methods presented in this paper have been successfully integrated into the OpenWSN platform, demonstrating their practical applicability. By providing a specific approach to handle heavy traffic in IIoT networks, this research offers valuable insights to enhance the performance of 6TiSCH networks.

In a study[26] led by Righetti et al., the primary focus is on evaluating the performance of the 6top protocol within the 6TiSCH architecture during the initial network formation, with a specific focus on the 6P transaction's role in resource negotiation. The research aims to optimize the allocation of resources during the initial network setup, focusing on key parameters, such as the number of 6top timeslots, the 6top timeout, and MacMinBE.

These parameters influence the success rate of 6P transactions, the allocation delay, and inconsistencies during network bootstrap. In scenarios with static routing, it was observed that an increased number of allocated timeslots significantly reduces unsuccessful scenarios, ensuring more effective message delivery. This is why they recommend allocating a minimum of two 6top timeslots is crucial to reduce congestion and enhance the success rate. The research also emphasizes the importance of configuring the timeout parameter to maximize the success rate of 6top transactions. According to their findings, a higher timeout indeed leads to a higher transaction rate, primarily because of the retransmissions of 6top messages. On the other hand, the study reveals that a shorter timeout results in more sequence number inconsistencies. Consequently, they recommend a timeout range of 40s to 60s to optimize the success rate of 6top transactions. Additionally, the study highlights that the MacMinBE parameter's influence on protocol performance is generally minor. However, setting MacMinBE to 4 significantly degrades performance. To minimize transaction delays, it is recommended to set MacMinBE to 1, as lower values result in slightly shorter transaction delays, promoting more immediate retransmissions following collisions. This research also explores the impact of the RPL routing protocol's dynamics on 6P transactions during network bootstrap. It reveals that frequent changes in the preferred parent, a feature of RPL, can lead to transaction failures. To address this, the study suggests the use of active probing and a more conservative RPL configuration to enhance the stability and success rate of 6P transactions during network bootstrap.

3.4. Novelty

After reviewing existing research in the field of our study, it is imperative to delve into the gaps and limitations inherent in previous studies. In this discussion, we will compare and contrast our investigation with these prior works, highlighting the distinctive focus and approach of our study. The majority of research in this field has primarily focused on evaluating and optimizing MSF or, more specifically, the 6top protocol. When studies have been conducted to assess the performance of the 6top protocol, their emphasis has been on identifying the causes and factors that lead to 6p transaction failures, or on finding ways to enhance the protocol to prevent such failures. However, what sets our research apart is our dedicated focus on exploring the impacts and consequences of 6p transaction failures on network performance. It's important to note that none of the existing research has delved deeply into the outcome of these failures or assessed their consequences in terms of various KPIs. Previous studies either briefly mentioned these impacts or relied on their expectations rather than conducting a comprehensive evaluation.

Even in the research efforts that aimed to evaluate the performance of the 6top protocol, the scope was generally limited to setting up experiments to measure parameters such as 6p failure rates, transaction delays, and 6P transaction overhead. However, none of them directly assessed the impacts of these failures after they occurred. Our study seeks

to bridge this substantial gap in the existing research by comprehensively investigating possible aspects of transaction failures, particularly the consequences of 6p transaction failures on network performance. While many prior studies relied on simulations, which offer flexibility but may not precisely replicate real-world conditions and challenges, our approach is distinct. We conducted our experiments using physical devices, employing the OpenMote-B hardware and Contiki-NG as the operating system. This combination allowed us to generate results and evaluations that closely resemble real-world conditions, capturing the nuances and complexities of practical scenarios.

In our experiments, we went a step further by artificially defining different failure probability ranges, spanning from low to high. This artificial manipulation enabled us to gain a deep understanding of how different failure scenarios impact network performance. Overall, our approach is geared towards providing a more realistic and comprehensive evaluation of the 6top protocol's performance, considering not only its ideal functionality but also its responses and adaptability in the face of transaction failures in a real-world context.

4. Methodology

4.1. Context

This chapter discusses the methods and approaches used in the research. As mentioned earlier, the 6TiSCH protocol stack is essential in ensuring the reliability and integrity of Wireless Sensor Networks. It does this by allocating communication links, known as cells, in a way that allows every network member to transmit and receive within designated time and frequency slots. This approach ensures smooth and uninterrupted communication without any congestion.

At the core of this mechanism is the scheduling function, which is responsible for making critical decisions about cell allocation. This allocation process requires the use of the 6P protocol. However, before communication cells can be allocated, nodes within the network must engage in negotiation. Successful negotiation results in the addition of new allocation cells to the scheduling table of neighboring nodes, enabling seamless communication. However, network protocols often face disruptions and fluctuations in link quality due to environmental conditions. In such scenarios, 6P transactions, which are essential to the functioning of the 6TiSCH stack, may experience interruptions, resulting in incomplete transactions. These disruptions can have significant implications and negatively impact network performance. The extent of these effects depends on various factors, including network topology, Network Size, 6P messaging type, and more.

The research aims to comprehensively investigate the performance of WSN networks in the event of 6P transaction failure. To assess and analyze the effects of such disruptions, various KPIs are meticulously examined. These KPIs serve as critical metrics for gauging network performance and reliability under different conditions. Throughout the remainder of this chapter, the methodologies employed to evaluate and analyze the impact of 6P transaction failure will be explored. Additionally, the discussion will include the specific KPIs investigated, as well as the tools and techniques employed in the research to gain a deeper understanding of the implications of 6P transaction failure on network performance

4.2. Methodological Approach

As discussed in the background chapter, the components of 6P transactions consist of requests, responses, and acknowledgments. Furthermore, the 6P request messages

encompass Add, Delete, Relocate, and Clear. The failure of any of these components results in an unsuccessful 6P transaction and may yield various consequences on the network's performance. Due to limitations in available implementations and time constraints in this thesis, we concentrated on the failure of responses to 6P Add requests to assess the impact of unsuccessful 6P transactions. The reason for this choice is that the Add request is the most critical for network performance, and it is commonly used in shaping TSCH schedules. Its failure can pose the most significant challenges compared to other types of 6P requests. Add request is triggered when sensor nodes face a lack of communication resources to handle increasing traffic, and that's why understanding the impact of its failure is crucial.

In addition to 6P Add requests, we also investigated the impact of clear messages, as schedule inconsistency is another crucial aspect. However, because clear messages occur less frequently, a smaller portion of our experiments covered this scenario. Details about the implementation and challenges we encountered, will be discussed chapter 5. In this thesis, understanding the effects of failure on other 6P messages relies more on hypotheses rather than direct experimental data.

Our approach to evaluating the impact of 6P transaction failure on network performance relies on a quantitative methodology, which involves collecting and analyzing numerical data through different Key Performance Indicators. To ensure the accuracy and relevance of our evaluations, we chose to use physical devices in our experimental setup. Specifically, we utilized OpenMote-B boards with the Contiki-NG operating system as our hardware and software platform. Our goal was to replicate real-world conditions as closely as possible, aligning with our research philosophy of conducting real-world-oriented investigations. While it is true that employing physical devices introduced its own unique challenges and limitations compared to simulation tools, it also brought the distinct advantage of producing results that closely approximate real-world conditions. Consequently, these results provide a more tangible and reliable basis for real-world comparisons. This combination of a quantitative research approach and a real-world-oriented experimental setup forms the core of our methodology for evaluating the impact of 6P transaction failure on network performance. In the upcoming subsections of this methodology chapter, we will provide a comprehensive discussion of the features, challenges, and specific details related to the utilization of OpenMote-B boards and the Contiki-NG operating system in our experimental setup.

4.3. General View of Experimental Setup

In this section, we will present a general schematic of our experimental design, which will provide a foundational understanding of our approach, helping to clarify our methods in future sections. It's important to note that while we provide an overview here, the detailed explanation of each component will be covered in subsequent sections and chapters, particularly in the implementation chapter. As a general view of our setup,

all our experiments are based on a linear topology consisting of two nodes: one acting as the sender and the other as the receiver. Between these nodes, there exists a UDP traffic flow from the sender to the receiver. The sender, based on its traffic needs (after sending a certain amount of UDP packets), triggers a 6P Add request to initiate a 2-step transaction aimed at adding a new cell to its schedule. Importantly, the response to each of these Add requests, based on predefined probabilities, may randomly drop. This introduces uncertainty, leading to one of two scenarios: either a successful transaction or a transaction failure.

With this general overview of our setup design, we conducted a series of experiments for each Key Performance Indicator to collect data. This data will be used for analysis to evaluate the impact of 6P transaction failure on the performance of the network.

4.4. Data collection

In our experimental process, we conducted all the experiments to gather data for our Key Performance Indicators during the Schedule adaptation period. For each experiment conducted, we repeated the process multiple times to gather a sufficient number of samples for analysis. Specifically, we conducted each experiment 10 times, resulting in a total of 10 samples for each data point. To assess the statistical significance and accuracy of our findings, we employed a confidence interval with a confidence level of 95%. This approach allowed us to estimate the range within which the true population parameter is likely to fall, providing a measure of the precision of our results. It's important to note that while we primarily used confidence intervals for our analysis, in the respective sections dedicated to each Key Performance Indicator, we will explore and explain any alternative statistical approaches, if applicable.

4.5. Contiki-NG

As mentioned, for our experimental setup, we employed the Contiki-NG operating system as the core software platform. Contiki-NG is a lightweight open-source operating system designed for IoT devices. It was released in November 2017 as a fork of the original Contiki-OS to overcome its limitations. Contiki-NG offers a reliable platform for energy-efficient communication and supports a range of IETF standard protocols, such as 6TiSCH, CoAP, RPL, and 6LoWPAN. It is compatible with a wide variety of hardware platforms, primarily focusing on Arm Cortex-M devices such as those utilizing Cortex-M3/M4, as well as the Texas Instruments MSP430. Contiki-NG's network stack includes the complete implementation of TSCH, 6TiSCH, and the 6top sublayer. Within the implemented 6top sublayer, there are two scheduling functions available, the Minimal Scheduling Function and Orchestra [38]. For our experiment, we used MSF as the default and baseline scheduling function. However, it should be noted that the Minimal scheduling function in

Contiki-NG, at the time of this thesis, did not completely meet the requirements for RFC 9033 [23]. Only a simplified version of MSF was available as an example, and we had to extend it to meet our research objectives. Later in this thesis, we will further discuss the improvements we made to enhance the basic MSF.

4.5.1. Cooperative Multitasking in Contiki-NG

Contiki-NG uses a model based on events that facilitate cooperative multitasking [39]. In Contiki-NG, numerous applications are created through Process abstraction. When an application is launched, a process will remain inactive until an event occurs. Upon arrival of an event, the process handles it by performing the necessary tasks and then pauses until the next event. It's important to note that Contiki-NG's scheduler functions cooperatively, which means that processes willingly hand back control to the scheduler once their tasks are completed. Unlike preemptive systems, the scheduler never forces task switches in this cooperative model. However, it's worth mentioning that hardware interrupt handlers still have the ability to interrupt process execution. we used process abstraction functionality of Contiki-NG to build our setup application. Further details about our application will be explained in subsequent sections.

4.5.2. 6top implementation in Contiki-NG

Exploring the 6top architecture developed in Contiki-NG provides valuable insights into the background operations. This knowledge is essential for making precise changes to the design and implementation of our setup. Therefore, in this section, we discuss the 6top architecture developed in Contiki-NG. The 6top sublayer implemented in Contiki-NG consists of 5 software modules, as represented in C source files. The functionality of each module is detailed in Table 4.1

Module	Description
sixtop	Provides external APIs to control 6top
sixp	Handles 6P incoming and outgoing messages
sixp-nbr	Manages 6P-related states of each neighbor
sixp-trans	Manages ongoing 6P transactions
sixp-pkt	Provides helpers for 6P message manipulation

Table 4.1.: 6top Software Modules in Contiki-NG [40].

Figure 4.1 illustrates the internal architecture of the 6P implementation in Contiki-NG, showcasing the interactions among the five modules. In this architecture, Sixtop serves as the interface for communication with external modules, including the TSCH layer and

scheduling function. The Sixp module, responsible for generating 6P transactions, encompasses two pivotal functions: sixp_output() and sixp_input(). Both of these functions play crucial roles in our experiment, as they constitute the core operations necessary for applying 6P transaction failure. When the scheduling function invokes sixp_output() to

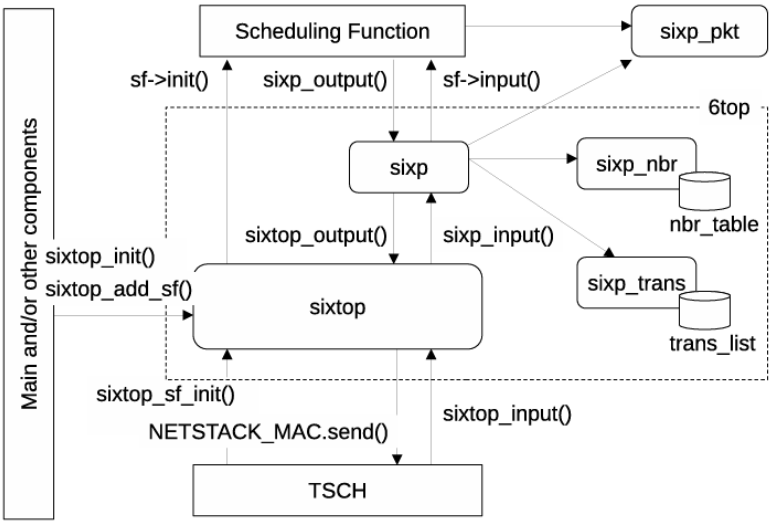


Figure 4.1.: 6P Architecture in Contiki-NG [40].

send a 6P message, four essential tasks must be performed by sixp_output(): creating a transaction state for requests, obtaining a sequence number through sixp_trans for use in transactions, constructing the 6P message, and ultimately transmitting the message [40]. Algorithm 1 (adapted from [40]) provides the pseudocode for sixp_output().

Algorithm 1 Pseudocode of sixp_output()

```
1:  procedure 6P MESSAGE TRANSMISSION
2:      if message is request then
3:          Create a transaction state
4:          Create/find a neighbor state
5:          Obtain a sequence number from the neighbor state
6:      else
7:          Find a transaction state
8:          Obtain a sequence number from the transaction state
9:          Build a 6P message
10:         Transmit the 6P message (call NETSTACK MAC.send)
```

On the other hand, sixp_input() (see Algorithm 2, adapted from [40]) is responsible for handling incoming requests. Upon receiving a 6P request, it first checks the Scheduling Function Identifier (SFID) of the scheduling function and verifies the sequence number

of the message. If there are no issues with the message, its body is then passed to the scheduling function for further processing.

Algorithm 2 Pseudocode of sixp_input()

```
1:  procedure 6P Message Reception
2:      Parse a 6P message
3:      Find a destination scheduling function
4:      if message is request then
5:          Create/find a neighbor state
6:          Create a transaction state
7:          Validate the sequence number against the neighbor state
8:      else
9:          Find a transaction state
10:         Validate the sequence number against the transaction state
11:         Pass body to the scheduling function (call its input)
```

4.5.3. Contiki-NG Timers

Timers are essential components in Contiki-NG, as they facilitate tasks such as timing events, detecting timeouts, and scheduling real-time operations. Contiki-NG offers multiple timer libraries and a clock module to manage time-related operations efficiently. All these timers build on the clock module, which is responsible for maintaining the basic system time. The Timer library offers a simple timer mechanism that does not include built-in notifications. This makes it a reliable choice for tracking elapsed time without immediate callbacks, as it is safe from interrupts. Stimer is an extension of Timer and measures time in seconds, making it suitable for longer time intervals. It is also safe from interrupts, which ensures stability for extended-duration operations. The Etimer library is designed to schedule events based on timers. It generates the *PROCESS_EVENT_TIMER* event when an Etimer expires and directs it to the associated process. Etimer is not safe from interrupts, which makes it ideal for event-driven tasks that require precise timing. Ctimer is similar to Etimer but schedules timers for callback function execution when they expire. Ctimers are not safe from interrupts, and they are used for executing specific actions at precise times. For real-time task scheduling with high clock resolution, the Rtimer library is the best choice. Real-time tasks scheduled through Rtimer can preempt normal execution, running immediately. Although they are safe from interrupts, they require careful handling due to potential conflicts with regular execution. Real-time tasks are essential for addressing time-critical operations efficiently.

Together, these timer libraries, in combination with the clock module, provide essential tools for managing time and scheduling tasks effectively in Contiki-NG. For our time duration and delay measurements, we rely on the clock module, particularly the

clock_time() function. This function allows us to capture the current system time in CPU clock cycles. In our experiments, we often initiate the timer with start_time = clock_time() at the beginning of a specific operation. Then, at the conclusion of the operation, we record the end time with end_time = clock_time(). These values are expressed in ticks, representing the number of clock cycles that have occurred since the system's inception. However, to provide more practical and meaningful measurements, especially when dealing with real-world time intervals, we need to convert these CPU clock cycle counts into seconds. This conversion involves considering the system parameter CLOCK_SECOND, which represents the number of ticks per second and is set to 128 in Contiki-NG for the OpenMote-B platform. Using the following formula:

$$\text{Time in seconds} = \frac{\text{end time} - \text{start time}}{\text{CLOCK\_SECOND}} \tag{4.1}$$

This conversion enables us to express time durations and delays in seconds.

4.6. OpenMote-B

We have selected OpenMote-B as our hardware for our experiments. This platform is specifically designed for hardware development and prototyping within the Industrial Internet of Things domain. It is especially useful for researchers and developers focused on advancing the field of long-range and low-power wireless field area networks. The OpenMote B, which works with the IPv6 stack, offers key features that make it an ideal choice for our research. A key feature of the OpenMote-B is its utilization of the Texas Instruments ARM Cortex-M3 based Ti CC2538 System on Chip (SoC), providing 512KB of Flash memory and 32KB of RAM. Additionally, the platform is equipped with the At-mel AT86RF215 radio transceiver. Another notable feature is its simultaneous multi-band operation capabilities, allowing it to operate seamlessly across the 2.4 GHz and 868/915MHz ISM (Industrial, Scientific, and Medical) bands. This feature is particularly useful when working with diverse communication frequencies in the IoT realm. The OpenMote-B is fully aligned with the latest IEEE 802.15.4 standards, ensuring complete compatibility with contemporary IoT protocols and networking standards.

Figure 4.2.: The OpenMote-B hardware platform.

4.7. Hypothesized Outcomes of 6P Transaction Failure

Before we discuss the KPIs used in our study, it's important to understand the potential impacts of 6P transaction failure. These impacts are based on our hypotheses and help us to provide insights into our selected KPIs. Our aim is to clarify the role of these key performance indicators and their importance in the context of 6P transaction failure. By exploring these impacts, we hope to shed light on the expected outcomes of the KPIs that will be discussed in the next section.

There are different types of 6P messages (requests, responses, and confirmations), and the consequences and effects can vary depending on each type of 6P message. Additionally, various scheduling functions can be employed in the 6TiSCH protocol stack, and based on their behaviors in handling 6P transaction failure, we anticipate different impacts on network performance. In this thesis, our primary focus is on the minimal scheduling function as the default, and we will discuss how it behaves in response to errors caused by 6P transaction failure. To explore the possible consequences of failing 6P transactions, we hypothesize potential outcomes in two cases: first, the case when a Failed 6P Transaction is followed by Success, and second, the worst-case scenario when a 6P Transaction Failure leads to a schedule reset.

4.7.1. Failed 6P Transaction Followed by Success

In an ideal and error-free 6P transaction scenario, let's assume that Node A initiates an Add request transaction to allocate a cell based on its traffic needs. Node B is expected to reply with a SUCCESS message when everything goes fine. However, if the SUCCESS response message becomes lost in transit, Node A must perform a retransmission of the Add request when the Timeout is reached. In this case, the transaction eventually succeeds after some retransmissions and a certain number of timeouts. These retransmission and timeouts cause delays and can negatively affect the overall performance of the network. Some of the possible impacts are as follows:

Increased Latency:

This effect's most obvious and immediate side effect is increased latency. Retransmissions and timeouts cause delays in the transaction process, delaying the transmission of data packets. For time-sensitive applications where minimal latency is essential, such as industrial automation or real-time monitoring, this can be especially challenging.

Reduced Throughput:

The network's overall throughput, or data transfer rate, may drop as a result of delays. This is due to the fact that the retransmission attempts consume valuable timeslots that could have been used for data transmission.

Energy Inefficiency:

The increased communication activity associated with retries and timeouts can lead to higher energy consumption, particularly in battery-powered devices. This can shorten the battery life of nodes in the network.

Impact on Real-Time Applications :

Applications that rely on real-time data, such as remote control systems, may experience interruptions or degradation in quality due to increased latency and delays. As retries and timeouts occur, they can contribute to network congestion, especially in high-traffic scenarios. This congestion can lead to packet collisions and an increased likelihood of packet loss.

Resource Allocation :

we know that the minimal scheduling function decides to add a new cell when it determines that cell usage exceeds the predefined High threshold. It means that a node is experiencing high traffic demand, and there is a need for a new cell to accommodate this increased communication load. failure to add a cell can lead to suboptimal resource allocation. Without the additional cell to accommodate the growing traffic demand, the existing cells may become overloaded, leading to congestion and packet loss within the network. These issues can have a cascading effect, resulting in the degradation of overall network performance and reliability.

When a node initiates a 6P transaction, the involved cells are locked until a response is received, preventing conflicting operations. The impact of this locking mechanism lies in the temporary unavailability of scheduled cells for other communication needs, as concurrent transactions or communication requirements involving the locked cells cannot proceed until the transaction completes. If a node lacks resources for concurrent transactions, it responds with an RC_ERR_BUSY code. Additionally, if requested cells are already locked, the node responds with an RC_ERR_LOCKED code, necessitating retry mechanisms for the affected nodes to regain access to the locked cells. This retry leads to potential delays and increased contention for network resources.

Resource Waste :

When the scheduling function determines that cell usage is low, it decides to initiate cell deletion. In the case of a failed Delete request, nodes may encounter difficulties in efficiently removing unnecessary cells from their schedules. The inability to remove these unneeded cells leads to resource inefficiency since the cells remain allocated even when they are not actively used. This inefficient resource allocation can result in an overallocation of communication resources within the network. Unused cells tie up bandwidth and may prevent other nodes from utilizing the schedule effectively. This overallocation not only

impacts resource utilization but also increases energy consumption. Since the unnecessary cells are still maintained, they consume energy unnecessarily, thus decreasing the overall energy efficiency of the network.

Relocation Inability

As explained in Chapter 2, if the packet delivery ratio (PDR) of each cell falls below the predefined value of RELOCATE_PDRTHRES (typically set to the default threshold of 50%) due to persistent congestion issues, the scheduling function initiates a 6P relocate message to adjust the cell's location within the schedule. This relocation aims to resolve congestion problems and improve network performance. However, if the relocate message encounters a failure, the cell relocation process becomes unsuccessful. As a result, the cell continues to suffer from congestion and interference, which can significantly impact network performance and reliability, leading to potential data loss and decreased overall data reliability.

4.7.2. Worst-Case 6P Transaction Failure with Schedule Reset

In the worst-case scenario, 6P transactions may repeatedly fail, often due to persistent message corruption or loss. After multiple retries and timeouts without success, the network may, in some cases, detect sequence number inconsistency, as mentioned earlier. In this situation, the Scheduling function, as per MSF guidelines, faces the decision of either clearing the schedules of both neighboring nodes through a 6P CLEAR request or attempting to retrieve the schedule using a 6P LIST request. Both of these options result in a significant overhead for the network.

Clearing the schedule involves canceling all previously allocated cells, and rebuilding the schedule from scratch can be time-consuming and resource-intensive. On the other hand, retrieving the schedule requires collecting information about all allocated cells in the network, which can also be resource-intensive. Additionally, scheduling functions like MSF often permit only a single transaction at a time, which further complicates the process. In either case, addressing 6P packet loss that leads to sequence number inconsistency results in a significant overhead.

The hypotheses presented in this section serve as a valuable framework for our subsequent analyses. These hypotheses will guide our selection and examination of key performance indicators in next section.

4.8. Key Performance Indicators

In this section, we outline the metrics we used to measure the influence of 6P transaction failures on network performance. Through a series of experiments, we aimed to understand

how 6P packet loss delays data transmission and disrupts network behavior. To achieve this, we relied on Key Performance Indicators essential metrics for measuring and analyzing the effects of 6P transaction failures. These metrics include end-to-end delays, Packet Delivery Ratio, Schedule adaptation duration, and energy consumption. This section offers a detailed overview of our chosen KPIs for this evaluation.

In the subsequent section, we will delve into the implementation details of our experiments, providing a comprehensive understanding of our methodology. Following that, in Section 5 (Analysis), we will analyze our results in light of these KPIs to draw meaningful conclusions about the impact of 6P transaction failures on network performance.

4.8.1. Schedule Adaptation Duration

Adapting to traffic is a crucial task of the Minimal Scheduling Function to ensure efficient resource allocation and optimal network performance. A node, when faced with increasing traffic demand, must be able to dynamically adjust its communication schedule. It achieves this by issuing a 6P Add request to its neighbor to allocate new cells, ensuring the network can accommodate the growing demand effectively. This schedule adaptation, by increasing the number of cells, continues until a point where the cell usage ratio falls below its high threshold (within the range defined by low and high thresholds). We can say that Minimal Scheduling keeps expanding the schedule until the cell usage reaches an acceptable level, where current cells in the schedule can handle high traffic demand without congestion.

In situations where traffic is high, it becomes important that schedule adaptation be done seamlessly. Any disruption or delay can extend this adaptation duration, which can potentially impact the overall performance of the network. In our study "Schedule Adaptation Duration" refers to the time it takes for a node to adjust its communication schedule by incorporating a designated number (N) of cells into its schedule. In our designed setup, Node A periodically initiates a 6P transaction to add a cell after sending a certain number of UDP packets. This procedure is repeated until the predefined maximum number of cells required for the schedule is reached. We have to note that in our implementations, monitoring bandwidth for tracking cell usage is defined by sending UDP windows. The reasons for this choice will be discussed in detail in Chapter 5.

One of the disruptions that can extend the duration of schedule adaptation could be the occurrence of a failure in the 6P Add transaction. This will cause a delay in the procedure of schedule adaptation. That's why this metric is valuable to be considered as one of the key performance indicators, assessing the impacts of 6P transaction failures on the network's performance.

To comprehensively assess the influence of these delays, we conducted the experiment with varying probabilities (P) of 6P response loss, ranging from 0.1 to 0.9. This wide range of probabilities highlights the varying degrees of uncertainty and unreliability that nodes may encounter during schedule adaptation. Furthermore, we explored the experiment

with different desired maximum numbers of cells in the schedule (N = 5, 10, 15, 20) to understand how the interaction between the choice of timeout value (T*timeout*: 10 seconds and 20 seconds), the probability of 6P response loss, and the desired schedule size collectively impact the "Schedule Adaptation Duration." The time units for schedule adaptation in our experiments were collected in milliseconds but were converted to seconds for representation in our analysis. The reason for this choice is that milliseconds would yield too high a resolution, making it difficult to comprehend and understand time durations in cases with long schedule lengths where loss probability is high. In such cases, the adaptation process can even take nearly hours, and representing these durations in milliseconds might result in excessively large values. Converting and representing time in seconds provides a more suitable scale for a clearer interpretation of the results.

To track the time it takes for schedule adaptation, we started the clock timer when the Add request for adding the first cell was issued by the sender. Then, we stopped the clock timer when the 6P transaction of the last cell was successfully completed and last cell added to schedule.

Analytical Model for Schedule Adaptation Duration :

In addition to our experimental observations, we also developed a mathematical estimation to predict the total time duration of schedule adaptation. Our mathematical model takes into account the chosen probability (P) of 6P response loss, providing a predictive tool to estimate the delay of such uncertainties in the schedule adaptation process. This estimation is expressed by equation 4.2 :

$$T_{\text{SA}} = N \cdot [E[R] \cdot ((T_{\text{udp}} + T_{\text{fail}} + T_{\text{timeout}}) + T_{\text{success}})] \tag{4.2}$$

Let's break down the equation and its components:

- T_{SA} represents the Estimated Schedule Adaptation Time, which is the total time required for a node to adapt its communication schedule.
- N is the maximum number of cells that need to be added to the schedule. It represents the size of the schedule that the node aims to achieve.
- E[R] denotes the Average Number of retires. This value is dependent on the 6P response loss probability (P), and it quantifies the number of retries a node may need to perform to successfully complete a 6P transaction. It is given by

  $$E[R] = \frac{1}{1 - P} \tag{4.3}$$

  where P is the probability of 6P response loss.
- T_{udp} stands for the time required for transmitting a predefined number of UDP packets (in our experiment, it's set to 3). We measured its value to be 4946 ms.

- T_{fail} represents the time for the transmission of a single Add request, which ultimately result in response loss. Based on our measurement, this time is 7 ms, approximately equivalent to one tick.
- T_{timeout} is the timeout value set for 6P transactions. This can be either 10 seconds or 20 seconds, depending on your experimental setup. It is a critical factor because transactions that don't succeed within the timeout duration may lead to retries.
- T_{success} denotes the time required for a single transaction to be successfully completed. Its recorded value is 130 ms based on our measurements. This duration represents the ideal scenario when a 6P transaction completes without any retries or timeouts.

  We can interpret Equation 4.2 in two scenarios:

1. **Error-Free Transaction:**

   If a transaction is error-free (i.e., it doesn't face response failures), then what we get from the equation is $T_{\text{udp}} + T_{\text{success}}$. This component represents the time needed to transmit a predefined number of UDP packets and the time for a single transaction to successfully complete. When multiplied by N (the desired number of cells), it gives us the duration for schedule adaptation without any failure in 6P transactions.

2. **Transaction with Failures:**

   However, in cases where transactions face failure due to 6P response loss, the equation takes number of retries and the timeout value into account , which leads to an increased time duration for adapting the schedule with the desired number of cells.

   This analytical model enables us to estimate schedule adaptation duration and understand how various factors, such as 6P response loss probability, timeout values, and desired schedule size, impact the overall adaptation process.

Expected Outcome :

We expect a significant increase in the duration of schedule adaptation by increasing three key parameters: the loss probability, the number of cells, and the timeout value. As these parameters are associated with greater uncertainty, larger scale scheduling, and more extended response waiting times, we expect the "Schedule Adaptation Duration" to reflect these complexities, leading to longer durations for the schedule adaptation process.

4.8.2. End to End Delay

Another key performance indicator that we considered in our study is measuring End-to-End Delay to assess how 6P response failure impacts this critical metric. End-to-end delay, in the context of our study, refers to the time it takes for a data packet to traverse the entire network, from the sender to the receiver. The process of adding a new cell to

the communication schedule is driven by the need to address network traffic requirements efficiently. However, when the 6P Add request transaction encounters a failure, particularly in the case of the response message being lost, the allocation of a new cell can experience delays. This delay is significant, especially in scenarios with high network traffic patterns. In such situations, incoming packets may be queued for an extended period, awaiting their turn for transmission. The delay in cell allocation results in a backlog of packets, and they cannot be served promptly due to the lack of available transmission cells. Consequently, this delay in packet transmission significantly contributes to the End-to-End Delay for each packet.

Measurement of end-to-end delay in our study is conducted during schedule adaptation. in our designed experiment the sender generates packets at a rate represented by the parameter λ, which is set to be equal to N-1 packets per slotframe. where N is the maximum number of cells in the schedule that aims to be allocated. This means that, for each slotframe duration, the sender generates N-1 packets, simulating a high demand for network resources. This configuration mimics scenarios where the network operates close to its maximum capacity. This helps us evaluate how the network performs under conditions that mirror scenarios requiring overprovisioning, where network resources are allocated to handle near-maximum demand levels.

To accurately measure End-to-End Delay, we included the current system time in the packet payload at the sender node when a packet was sent. Upon reception of the packet at the receiver side, we extracted the timestamp. By calculating the time difference between the reception timestamp and the initial transmission timestamp, we were able to precisely quantify the duration of the End-to-End Delay for each packet.

Like the previous KPI we discussed earlier, for this experiment, we also repeated our experiment for different numbers of cells (N = 5,10,15,20) and two timeout values (10s and 20s), considering the probability of 6P response loss ranging from 0.1 to 0.9. This approach allowed us to explore how increasing the probability of 6P response failure affects the End-to-End Delay across various network configurations. By systematically varying the parameters, we gained valuable insights into the relationship between 6P response failure, schedule size, and the chosen timeout values, offering a comprehensive perspective on the impact of these factors on End-to-End Delay in our experiment.

Expected Outcome :

As the probability of 6P response loss increases, we can expect to see a corresponding increase in End-to-End Delay. This is because a higher probability of response loss means that more 6P transactions will fail, leading to delays in cell allocation and, subsequently, packet transmission.

4.8.3. Packet Delivery Ratio

Packet Delivery Ratio is another metric we evaluated in our experiments. This metric calculates the ratio of data packets that reach their intended destination successfully to the total number of packets transmitted across the network. It serves as a crucial indicator of the reliability and effectiveness of data transmission. By measuring the PDR, we can assess the frequency of successful data packet deliveries, even in the presence of challenges posed by 6P transaction failures, especially in scenarios where 6P response messages experience loss or delay.

To measure the PDR in our experiments, we employ a unique identifier included in each packet's payload during transmission. This identifier is subsequently extracted from received packets at the receiver. PDR is computed by dividing the number of received packets with matching identifiers by the total number of packets sent.

In our study, we specifically measured PDR for a single, well-defined scenario with the following parameter set: N=10, sending rate (λ) = 9 packet per slotframe, a timeout value of 10 seconds, and a range of 6P response loss probabilities from 0.1 to 0.9. This focused approach allowed us to demonstrate the general impact of 6P response failure on PDR. The results obtained from this single scenario can be extrapolated to a larger number of cells or even different timeout values, providing valuable insights into the broader implications of 6P transaction failures on the network's packet delivery performance. These insights will be further discussed in the analysis chapter of our study, where we will delve deeper into the implications and significance of these findings.

Analytical Model for PDR :

As a part of our study, we have developed a mathematical model to analyze PDR along with experimental investigations. In this section, we will provide an overview of our analytical model for PDR. Two important factors that can affect PDR are the number of available transmission cells in the schedule and the capacity of the Tx queue. While queue capacity remains a constant variable, the number of Tx cells can vary depending on traffic requirements. We acknowledge that transaction failures can lead to delays in resource allocation, potentially impacting the allocation of Tx cells and, consequently, the PDR. Our objective is to explore and understand how occurrences of failure in 6P transactions affect PDR. By deriving a mathematical equation to measure PDR, taking into consideration the number of retransmissions and timeouts due to 6P transaction failures, we are able to quantify packet losses and their impact on PDR.

In MSF, the MAX_NUMCELLS window is defined as the decision point where the node decides to adapt its schedule by adding or deleting cells. In our setup, this point is reached when our predefined number of UDP packets has been sent to the receiver. This means that the sender sends a certain amount of UDP packets, and then it decides to add a new cell. Incremental addition of cells starts from where we initially have an Autocell

for transmitting incoming packets, and it continues until we reach the maximum desired number of cells.

During this increasing trend, the service rate naturally varies, and by adding more cells in each round, the service rate improves. By 'round' we mean the time period when a cell is added to the schedule until the next decision is made. In the first round, there is only one Tx cell available to handle λ incoming packets. In this case, the service rate is represented by $\mu_i = 1$. This service rate increases over time as more Tx cells are added to the schedule, reaching its peak in the last round, where we achieve a service rate of μ_M = N. By considering the time it takes for the service rate to transition from μ_i to reach μ_M, we can calculate how many packets have been served and how many of them are lost. This allows us to calculate the PDR.

As each round passes and more Tx cells are added to the schedule, the number of elapsed slotframes required to transmit the packets decreases. To better understand this concept, let's examine an example provided in Figure 4.3 In this scenario, the incoming packet rate is $\lambda = 4$, and the maximum number of cells needed to be added to the schedule is N = 5, with gray cells indicating used cells. In the first round, there is one cell available to transmit 4 incoming packets. Therefore, 4 slotframes have to be passed until all 4 packets are served. After transmitting all 4 packets, it's decision time to add a new cell.

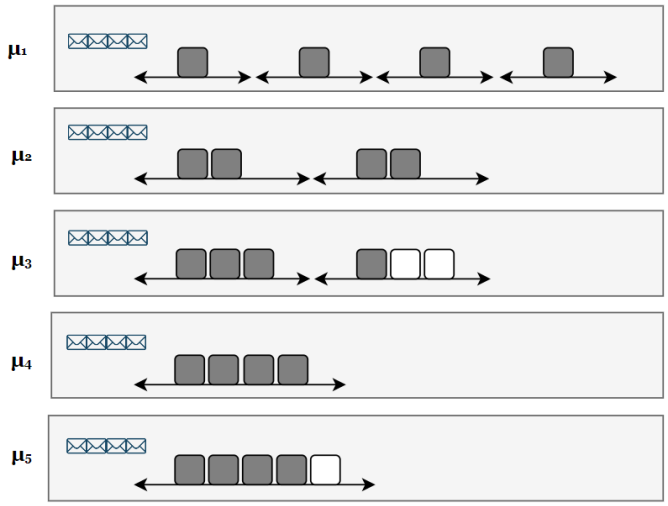


Figure 4.3.: Example for Service Rate transition from μ_1 to μ_5.

In the second round, there are 2 Tx cells available in schedule, so fewer slotframes are needed to serve the 4 packets. This pattern continues, and by increasing the round number, the number of slotframes needed decreases. The fraction of the decision window over the service rate ($\left\lceil \frac{W_{\text{udp}}}{\mu_i} \right\rceil$) gives us the number of elapsed slotframes in each round.

By summing up number of slotframes elapsed in all rounds ($\sum_{\mu_i=1}^{\mu_M}$) and multiplying it by the duration of the slotframe (T_{sf}), we are able to calculate the time duration required to transition from the actual service rate to the target service rate ($T_{\mu_i \to \mu_M}$). However, it's important to note that in our calculation, we must consider the occurrence of 6p transaction failure. Based on a 6p response loss probability range of 0.1 to 0.9, we may have retries (E[R]) and a timeout value (T_{timeout}) that come into play during the growing phase from μ_i to μ_M. This calculation is represented by equation 4.4 :

$$T_{\mu_i \to \mu_M} = T_{sf} \sum_{\mu_i=1}^{\mu_M} \left(\lceil \frac{W_{\text{udp}}}{\mu_i} \rceil \cdot E[R] + (E[R] - 1) \cdot T_{\text{timeout}} \right) \tag{4.4}$$

Where:

- $T_{\mu_i \to \mu_M}$: Time duration to transition from actual service rate to target service rate.
- μ_i : Service rate in i_{th} round.
- T_{sf} : Duration of each slotframe.
- W_{udp} : UDP sending window.
- $E[R]$: Average number of retries.
- T_{timeout} : Timeout value.

As sending rate is λ packets per slotframe, we can easily calculate the total number of packets sent (P_{total}), This is achieved by multiplying the number of slotframes passed in all rounds by the sending packet rate (λ), as shown in Equation 4.5 :

$$P_{\text{total}} = \lambda \sum_{\mu_i=1}^{\mu_M} \left(\lceil \frac{W_{\text{udp}}}{\mu_i} \rceil \cdot E[R] + (E[R] - 1) \cdot T_{\text{timeout}} \right) \tag{4.5}$$

To be able to calculate the Packet Delivery Ratio, in addition to knowing the total number of sent packets derived from Equation 4.5, we also need to determine how many packets have been lost during the transmission process. The calculation of the number of packets lost is influenced by the capacity of the Tx queue, which plays an important role in this context. Depending on the size of the Tx queue, the number of packet losses can vary significantly. For instance, in a spacious queue, more packets can be enqueued and wait for transmission, which may lead to a different loss pattern compared to a queue with limited capacity. However, before we take queue capacity into account, we need to understand how to calculate the number of packets that have arrived but have not been served in a single slotframe within each round.

The calculation performed for each round, based on the difference between the arriving packet rate per slotframe and the specific round's service rate of transmitting packets (λ-μ_i), enables us to establish how many packets were not served within that particular

round. When we sum up the results of all rounds, we can calculate the total number of remaining packets when we reach the last round, which is characterized by μ_M (Equation 4.6). If we subtract this cumulative number from the capacity of the Tx queue (q), we obtain the total number of packet losses (see Equation 4.7)

$$L = \sum_{\mu_i=1}^{\mu_M} (\lambda - \mu_i) \left(\lceil \frac{W_{\text{udp}}}{\mu_i} \rceil \cdot E[R] + (E[R] - 1) \cdot T_{\text{timeout}} \right) \tag{4.6}$$

$$P_{\text{lost}} = L - q \quad \text{where} \quad q = QueueCapacity \tag{4.7}$$

$$\text{PDR} = 1 - \frac{P_{\text{lost}}}{P_{\text{total}}} \tag{4.8}$$

Now, by knowing the total number of packet arrivals from Equation 4.5 and the total number of packet losses from Equation 4.7, we can calculate the Packet Delivery Ratio using equation 4.8. As mentioned earlier, PDR varies based on the Tx queue capacity. By utilizing these equations, we have calculated the PDR for different capacities to evaluate its performance under various scenarios. The analysis of these results is presented in Chapter 6.

Expected Outcome :

We expect that When 6P response failures occur, they introduce delays in schedule adaptation and Tx cell allocation. As a direct result, the lower rounds experience a shortage of available Tx cells, limiting their capacity to serve packets per slotframe. Meanwhile, incoming packets continue to arrive in each slotframe. This creates an imbalance between arrivals and service capacity. Consequently, it takes more slotframes to transmit the same number of packets, leading to an increase in the overall time required for packet transmission. With the limited service rate during these rounds, more packets will accumulate in the queue, contributing to an increased count of lost packets and consequently a decrease in the Packet Delivery Ratio.

4.8.4. Energy consumption

One of the KPIs that we are using to evaluate the impact of 6P transaction failures is energy consumption. In our experiment, we are particularly interested in understanding how energy is consumed when a node faces a 6P transaction failure. To accomplish this, we are leveraging the Energest module within Contiki-NG. The Energest module in Contiki-NG is a powerful tool for tracking and estimating energy consumption in resource-constrained IoT devices. It works by monitoring and recording the time that different hardware components are in various states (e.g., CPU active, low-power mode, transmission mode).

Parameter	Value
Voltage	3 volts
CPU Current	13mA
Transmit Current	24mA (Active-Mode Tx)

Table 4.2.: Power Parameters

By coupling this time data with knowledge about the power consumption characteristics of these components, we can estimate energy consumption accurately. For our experiment, we have decided to focus on two critical aspects of energy consumption: the transmitter and CPU. In our particular setup, the sender periodically increments the number of cells to transmit packets. In this scenario, both Tx and CPU energy consumption are of utmost importance. In this experiment, our primary objective is to measure energy consumption during the scheduling adaptation process and assess how the scheduling adaptation procedure, when confronted with a 6P response failure, impacts energy consumption.

To demonstrate this effect, we conducted energy consumption measurements within a scheduling process involving N = 10 (number of cells) while varying the 6P response loss probabilities from 0.1 to 0.9. To track the time spent in each state (CPU and Transmitter), we followed a systematic process. We initiated the tracking process by setting the Tx and CPU types to "on" using:

```
1 ENERGEST_ON(ENERGEST_TYPE_CPU);
2 ENERGEST_ON(ENERGEST_TYPE_TRANSMIT);
```

This action initiated the monitoring of the respective components and started tracking time. When we wanted to conclude the measurement, we set the specified type to "off" using :

```
1 ENERGEST_OFF(ENERGEST_TYPE_CPU);
2 ENERGEST_OFF(ENERGEST_TYPE_TRANSMIT);
```

This step stopped the tracking and updated the total time. To ensure that the total time was updated correctly, we called energest_flush before measuring the total time. This step helped us guarantee the accuracy of our energy consumption measurements.

The energy consumption was calculated using the formula 4.9:

$$\textit{Energy (mJ)} = \textit{Time (seconds)} \times \textit{Voltage (Volts)} \times \textit{Current consumption (mA)} \quad (4.9)$$

The specific values for Voltage and Current for our setup were obtained from the device datasheet, as shown in Table 4.2.

By default, the Energest module in Contiki-NG uses the Rtimer Library clock as its time source. When working with the time data obtained from Energest, it's important to remember to convert it to real time. This conversion can be done using the formula 4.10:

$$Time\ (in\ seconds) = \frac{Energest\ Type\ Time}{ENERGEST\_SECOND} \tag{4.10}$$

In this equation *ENERGEST_SECOND* value represents the number of ticks per second from the Energest time source. In our openmote-b platform, where a "tick" is equivalent to a cpu cycle, the value is 128. This process allowed us to accurately estimate the energy consumption associated with CPU and Transmitter operations during 6P transaction failures in our experiment.

Expected outcome :

We expect to observe an increasing trend in energy consumption with higher loss probabilities. This is anticipated because higher loss probabilities will lead to more retransmissions and increased timeouts, both of which contribute to elevated energy usage.

4.9. Schedule Inconsistency

One of the introduced behaviors in RFC 9033, which MSF should follow for 6P error handling, is the 'clear' behavior. This behavior is initiated in response to two error codes: *RC_ERR_SEQNUM* and *RC_ERR_CELLLIST*.

RC_ERR_SEQNUM signifies a schedule inconsistency caused by a mismatch of sequence numbers, and *RC_ERR_CELLLIST* denotes a CellList error, indicating a problem with the list of cells provided in a transaction. In both cases, when a node detects these errors during a 6P transaction, it must abort the ongoing 6P transaction and then initiate a new 6P transaction with a Clear Request directed towards its neighboring node that is experiencing the inconsistency. Subsequently, both nodes involved in the communication should remove all cells scheduled locally and rebuild those cells again. Clearing and rebuilding are two challenging tasks that can cause delays and interruptions in communication, leading to degraded network performance. For this reason, similar to the 6P Add request, it is also worth evaluating the impact of schedule inconsistency as one of the challenging aspects of our study.

As mentioned earlier, schedule adaptation is a crucial metric in our study, playing a significant role in responding to high traffic demand. Due to its pivotal importance, almost all of our experiments rely on it. To evaluate the impact of schedule inconsistencies, we conducted an experiment focusing on schedule adaptation. However, before conducting

this experiment, it was necessary to simulate a condition which mimics the occurrence of schedule inconsistency. The idea was that during the process of adapting the schedule, we deliberately introduced instances where clear messages were randomly issued by sender instead of Add request. This simulation aimed to replicate conditions similar to those that might occur in the presence of schedule inconsistency. The goal was to comprehend how these simulated instances of schedule inconsistency would influence the duration of schedule adaptation.

However, it should be noted that the conditions and settings for this experiment differ from those used in the experiment related to schedule adaptation duration as the key performance indicator. In this experiment, unlike our previous approach, the decision point for adding cells no longer relies on sending UDP windows. Instead, we aimed to align our experiment more closely with a realistic approach. In this modified approach, the decision point is based on tracking used cells, where new cell allocation is determined by monitoring the number of used cells.

In the realistic approach introduced by RFC 9033, the MAX_NUM_Cell, which determines the adapting decision point, is based on the number of used cells. This is analogous to tracking the number of sent packets, as for sending each packet, one Tx cell must be utilized, reflecting the number of cells used. With this in mind, we modified our application setup to adapt to these new changes. In the implementation chapter, we will delve into more detail about the implementation and the challenges faced during the process. Following that, we will present the collected results for this experiment and interpret them in discussion part.

5. Implementation

In this Chapter, we will describe the setup and application design of the testbed that was used to conduct the experiments in this study. In the following subsections, we will delve deeper into the Application design and Topology, which covers the network's architecture and application specifics. Then, we will focus on Extending the simple MSF, which involves customizing the Minimal Scheduling Function. Finally, we will discuss Simulating 6P Transaction Failures, where we induce and analyze these failures.

5.1. Topology

For our experiments, we intentionally utilized a simple linear topology, consisting of only two nodes. This specific selection of a single-hub topology provided us with a controlled environment which allowed for a precise analysis of the effects caused by 6P transaction failure. By deliberately choosing this limited setup, we could directly attribute any observed effects to the variables we studied. It is crucial to emphasize that the knowledge and insights gained from this initial two-node investigation will serve as the foundation for extending our results to more extensive network topologies.

5.2. Application Design

Contiki-NG is a collection of modules and libraries that offer a wide range of functionalities and capabilities for developers. These include low-level hardware interactions and high-level networking protocols, which allow developers to create customizable and efficient applications. Furthermore, Contiki-NG provides simple examples that users can use to test different functionalities or as a starting point for developing more complex applications. These examples served as a foundation for building our setup application, in the following subsections, we explain how we used and modified them to achieve our objectives.

5.2.1. RPL-UDP Application

For our application design process, we decided to use the RPL-UDP example provided by Contiki-NG as a foundation. This example was chosen because it offered a straightforward way to establish connectivity between two nodes and allowed us to observe a simple traffic pattern between them, making it a valuable starting point for our Application design.

From there, we worked to customize and improve it to meet our specific needs. The RPL-UDP codebase demonstrates a self-contained network configuration utilizing the Routing Protocol for Low-Power and Lossy Networks (RPL) in conjunction with the User Datagram Protocol (UDP) for communication. Unlike traditional networks, this setup doesn't rely on an external border router. Within this network, there are two types of nodes: a DAG (Directed Acyclic Graph) root and DAG nodes. The DAG root also acts as a UDP server, playing a central role in the network, while the DAG nodes operate as UDP clients, communicating with the DAG root.

Periodically, the UDP clients dispatch UDP requests to the server. These requests carry a counter value as payload. Upon receipt of a request, the server, or DAG root, promptly crafts a response containing the same counter value. This response is then dispatched back to the originating client. In our experiments, we refer to the client as the sender (*sender.c*) and the server as the receiver (*receiver.c*), aligning with our experimental terminology.

5.2.2. RPL-TSCH Application

The RPL-UDP codebase satisfies the essential criteria for our experiments, however, it is insufficient on its own. We needed to use the 6top module in order to evaluate the effects of 6P transaction failure. We attempted to expand the RPL-UDP code by taking inspiration from the RPL-TSCH example that uses a simple scheduling function to demonstrate TSCH capability. This extension gives nodes the ability to use TSCH for scheduled communication in addition to maintaining UDP communication capabilities. RPL-TSCH example operates in the role of a regular RPL-TSCH node. In this example, the role of a TSCH coordinator can be assigned to a node by adjusting the value of the 'is_coordinator' variable to 1, and 0 to indicate a non-coordinator. Including the TSCH module in the Makefile enables TSCH in our application.

5.2.3. SF-SIMPLE

RPL-TSCH example also provides a simple scheduling function (sf-simple.c) which is responsible for the dynamic allocation and deallocation of cells within the TSCH schedule. Within this code, cells are represented by structures containing timeslot and channel offset information. However, it's important to note that the channel offset is always assigned a value of 0 within this particular code implementation. In other words, the channel offset remains constant and does not vary when cells are added. The process of choosing timeslots incorporates a randomized approach, facilitated by the '*random_rand()*' function provided by the Contiki-NG. This function generates pseudo-random integers, which are used to select a random timeslot offset within the valid range.The valid range is determined by '*TSCH_SCHEDULE_DEFAULT_LENGTH*', which represents the maximum number of available timeslots in the schedule.

The functions *'add_links_to_schedule'* and *'remove_links_to_schedule'* are crucial in managing cell addition and removal based on peer node requests. The *sf_simple_add_links* function initializes variables, generates time slots randomly, and checks link availability. Constructing a 6P Add Request packet, it utilizes the *sixp_output* function for transmission. Upon a successful response, the *add_response_sent_callback* executes, seamlessly updating the local schedule. Similarly, *remove_links_to_schedule* follows a parallel procedure for removing links.

SF-Simple enforces a scheduling policy where cells are added if all requested slots are available and deleted if all requested slots are in use, contributing to the overall robustness and stability of the network. The '*sf_simple_driver*' structure, a part of the Sixtop interface, defines the behavior and characteristics of the Simple Schedule Function. It includes an SF identifier, timer intervals, and callback functions for input and notifications as default.

5.2.4. Main Application (RPL-UDP-TSCH)

To create our experimental application, we merged the distinct RPL-UDP and RPL-TSCH applications into a single cohesive framework, which we refer to as RPL-UDP-TSCH. This application comprises two fundamental processes: '*udp_client_process*' and '*cell_update_process*', each contributing unique functionalities to our application (see Listing 5.1).

```
1 /*---------------------------------------------------------------*/
2 PROCESS(udp_client_process, "UDP Traffic Flow");
3 PROCESS(cell_update_process, "Add/Delete Cell");
4 AUTOSTART_PROCESSES(&udp_client_process, &cell_update_process);
5 /*---------------------------------------------------------------*/
```

Listing 5.1: process.

The *udp_client_process* handles the task of sending UDP requests at regular intervals and processing incoming UDP responses. This process ensures the continuous flow of data within the application. On the other hand, the *cell_update_process* takes on the responsibility of adding a cell with a random timeslot and channel offset of 0 after a certain number of UDP packets have been sent. This addition of a cell is performed by the sf-simple, which is part of the RPL-UDP-TSCH framework. Together, these processes and the sf-simple form the core of the RPL-UDP-TSCH application. The general procedure depicted in Figure 5.1 outlines the RPL-UDP-TSCH flow. This diagram illustrates the step-by-step process that our RPL-UDP-TSCH application follows to achieve its functionality.

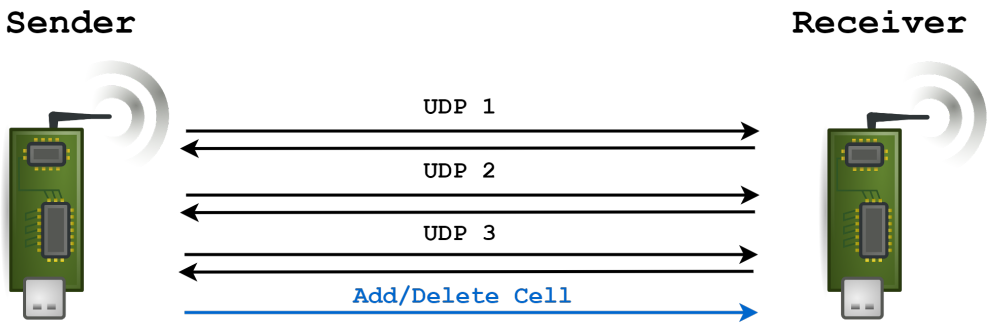


Figure 5.1.: RPL-UDP-TSCH flow with sending threshold of 3 UDP.

The fundamental steps in the communication process between a sender and receiver are outlined below :

1. Sender initializes the UDP connection and sets up a periodic timer for message transmission :

```
// Inside udp_client_process
simple_udp_register(&udp_conn, UDP_CLIENT_PORT, NULL, UDP_SERVER_PORT,
    udp_rx_callback);

etimer_set(&periodic_timer, random_rand() % SEND_INTERVAL);
```

2. Periodically sends UDP requests to the receiver node using the '*simple_udp_sendto*' function :

```
// Inside udp_client_process
simple_udp_sendto(&udp_conn, data, data_len, &dest_ipaddr);
```

3. Sender sets a flag ('*add_links_flag*') after reaching a certain number of message transmissions ('*Threshold_value*') :

```
// Inside udp_client_process
if (udp_tx_count >= Threshold_value) {
    add_links_flag = 1;
}
```

4. If the node is configured as a coordinator, starts the RPL root :

```
// Inside  cell_update_process
if (is_coordinator) {
    NETSTACK_ROUTING.root_start();
}
```

5. Sender activates MAC layer and integrates the SF Simple driver for link management:

```
// Inside  cell_update_process
NETSTACK_MAC.on();
sixtop_add_sf(&sf_simple_driver);
```

6. Sender adds links using the SF Simple driver in a time-slotted manner :

```
// Inside  cell_update_process
sf_simple_add_links(tsch_queue_get_nbr_address(n), num_links);
```

5.3. 6P Transaction failure simulation

In our experimental evaluation to assess the impact of 6P transaction failures on network performance, we introduced a controlled simulation of 6P response failures in response to 6P add requests. This simulation was conducted on the receiver side, enabling us to manipulate the probabilities of 6P response failures. These probabilities ranged from 0.1 (indicating a 10% failure rate) to 0.9 (representing a 90% failure rate).

The simulation was designed to mimic the unpredictability of real-world WSN networks, where packet loss can occur due to factors such as interference, congestion, or signal attenuation. By adjusting the failure probabilities, we could emulate different KPI levels, from relatively reliable to highly unreliable communication channels.

To implement this simulation, we extended the sixp_input() function to include a probabilistic response failure mechanism (see code snippet 5.2). When an incoming 6P request packet arrives at the sixp_input() function for processing, the simulation mechanism generates a random value. It then checks whether this random value is below a predefined threshold (e.g., 70% of RAND_MAX).

```
void sixp_input(const uint8_t *buf, uint16_t len, const linkaddr_t *src_addr
    )
{
    if (rand() < RAND_MAX * 0.7) {

        printf(" Packet loss occurred! \n");
        return;
    }
    .
    .
    // Rest of sixp_input() function
    .
    .
}
```

Listing 5.2: 6P Response failure

If the condition is met, signifying that the random value falls within the specified failure probability, the simulation considers the packet lost. Consequently, the simulation immediately returns from the sixp_input() function without further processing. In this case, no response is generated or sent in reply to that request.

After detailing our application setup and outlining the mechanism for generating 6P response loss, we will now visually depict the flow of our setup through a diagram (see Figure 5.2). This diagram will provide a comprehensive overview of the UDP communication flow and example of unsuccessful 6P transaction due to response loss.

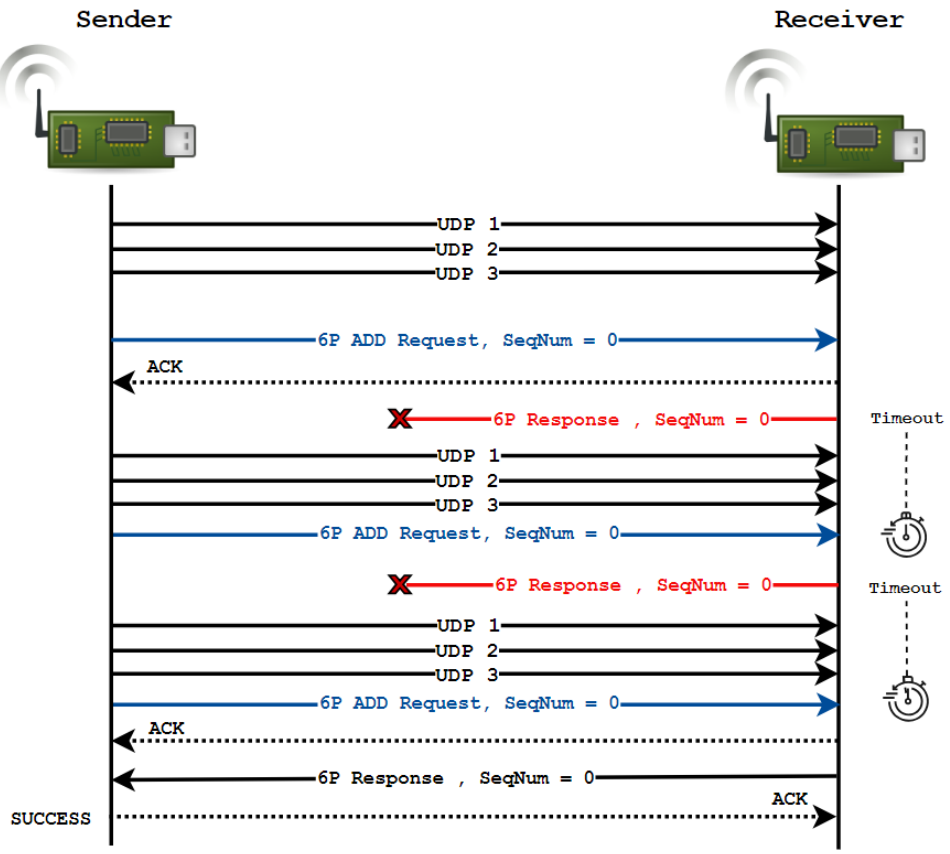


Figure 5.2.: example of 6P response failure with 2 losses.

In this scenario, the sender, after sending three UDP packets within a predefined sending window (detecting high cell usage), indicating a need for a new cell. In response to this, it initiates a 6P transaction by sending an Add request message with a sequence number of zero to the receiver. Upon successful reception of this request by the receiver, the response generation process is triggered. To generate the response and send it back to the sender, the receiver must invoke *'sixp_input()'*. However, the condition for the random loss generator within the *'sixp_input()'* function becomes true at the beginning, preventing the execution of the function body, causing an immediate return. Upon detecting this condition, the sender sets a timeout. During this timeout, the sender awaits a response

from the receiver while UDP traffic continues. After the timeout period elapses, the sender attempts the 6P transaction again by initiating a new Add request with the same sequence number as before. In this second attempt, the transaction is once again unsuccessful due to response loss. However, on the third attempt, the transaction becomes successful, allowing both the sender and receiver to add an agreed-upon cell to their schedules. This procedure continues until both the sender and receiver adapt their schedules and allocate the required number of cells to their respective schedules.

5.4. Transaction states

To gain a deeper insight into how artificially generated response loss impacts the flow of 6P transactions, it is beneficial to examine this scenario through the lens of transaction states. The transaction management system in Contiki-NG plays a crucial role in ensuring the smooth and efficient operation of 6P transactions. This system utilizes a defined set of transaction states to manage the entire process from initiation to termination. These states are instrumental in maintaining order and facilitating reliable communication between the sender and receiver. From the perspective of the sender, there are two distinct scenarios that arise :

Case 1: Successful Transaction

In this scenario, the 6P transaction proceeds without any failure, and the sender moves through various states to achieve a successful outcome. The sender's states are enumerated as follows:

- **State 1: INIT**

  The transaction begins in the INIT state, where it is initialized and prepared for execution.

- **State 2: REQUEST_SENDING**

  The sender advances to the REQUEST_SENDING state, where it transmits a request to the receiver. This request typically includes information about the operation to be performed.

- **State 3: REQUEST_SENT**

  After successfully sending the request, the sender progresses to the REQUEST_SENT state. Here, it awaits a response from the receiver, confirming the receipt of the request.

- **State 7: RESPONSE_RECEIVED**

  Upon receiving a response from the receiver, the sender enters the RESPONSE_RECEIVED state. This phase indicates that the request has been acknowledged, and the transaction is moving forward.

- **State 11: TERMINATING**

  Finally, in the TERMINATING state, the sender finalizes the transaction, ensuring that all necessary steps have been completed successfully. At this point, the transaction is ready to conclude.

Case 2: Transaction Failure

In this case, response loss leads to transaction failure. In such a situation, the sender recognizes that it is stuck in the RESPONSE_RECEIVED state (State 7) and has not received an acknowledgment from the receiver. To handle this situation, the sender terminates its state, initiates a timeout, and waits for a response. If no new response is received during this timeout period, the sender decide to start a new 6P transaction from the INIT state, as it cannot proceed with the current transaction. This case's sender states are as follows:

- **State 1: INIT**
- **State 2: REQUEST_SENDING**
- **State 3: REQUEST_SENT**

  After sending the request, the sender reaches the REQUEST_SENT state, expecting a response from the receiver. In the event that no response is received within the expected timeframe, the sender decides to terminate the transaction.

- **State 11: TERMINATING**

The entire transaction process, as viewed from both the sender's and receiver's perspectives, is illustrated in Figure 5.3.

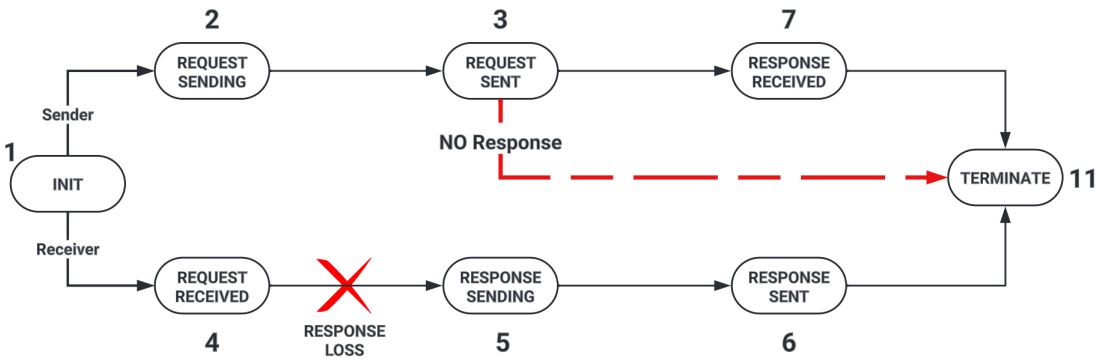


Figure 5.3.: 6P Transaction states.

5.5. Schedule Inconsistency

In the previous chapter, we clarified that, in addition to experiments related to key performance indicators, we also conducted an experiment to evaluate the impact of schedule inconsistency on the duration of schedule adaptation. We briefly explained that in this experiment, our approach for determining the decision point for adding cells was implemented based on tracking the number of used cells to align our experiment more closely with a realistic approach. It is important to note that this experiment was conducted as our final experiment, after all other main experiments and their results were collected. Due to this sequencing, we decided, according to the remaining time , to specifically rely on used cells as the decision point in this experiment. In this section, we will delve into the implementation details of this specific experiment.

5.5.1. Inconsistency Simulation

To assess the impact of schedule inconsistency on adaptation time, we introduced a method to simulate the occurrence of schedule inconsistencies during schedule adaptation. This simulation involved sending randomized clear messages from the sender to the receiver. To generate these randomized messages, we utilized the exact mechanism described in section 5.3. However, in this case, instead of simulating response loss, we introduced randomization in the generation of clear messages. The random clear message generator was integrated into the scheduling function at the entry point of the *'sf_simple_add_links'* function. This function is responsible for adding cells to the schedule. As outlined in the pseudo code (5.3), when the *'sf_simple_add_links'* function is invoked by the sender, the random condition is checked.

```
1  int sf_simple_add_links(linkaddr_t *peer_addr, uint8_t num_links) {
2
3      if (rand() < RAND_MAX * 0.7) {
4
5          assert(sixp_pkt_set_metadata(SIXP_PKT_TYPE_REQUEST,
6                                       (sixp_pkt_code_t)(uint8_t)
7                                       SIXP_PKT_CMD_CLEAR,
8                                       sample_metadata, req_storage,
9                                       sizeof(req_storage)) == 0);
10
11         assert(sixp_output(SIXP_PKT_TYPE_REQUEST,
12                            (sixp_pkt_code_t)(uint8_t)SIXP_PKT_CMD_CLEAR,
13                            SF_SIMPLE_SFID, req_storage,
14                            sizeof(sixp_pkt_metadata_t),
15                            peer_addr,
16                            NULL, NULL, 0) == 0);
17     }
18
19     // Rest of the function
20     // ...
21 }
```

Listing 5.3: Randomized Clear message generation.

If condition evaluates to true, the associated code block is executed, leading to the creation and transmission of a SIXP request packet. The packet is crafted using the *sixp_pkt_set_metadata* function, specifying the type as *'SIXP_PKT_TYPE_REQUEST'* and the command as *SIXP_PKT_CMD_CLEAR*. This process involves setting metadata and other necessary parameters. Following this, the constructed SIXP request packet is sent using the *'sixp_output'* function. The destination of this packet is determined by the peer address specified in *'peer_addr'*.

in the case that random condition evaluetes to false, indicating that the generation of clear messages did not occur, the body of the *'sf_simple_add_links'* function will be executed. In this scenario, the function will proceed with its default behavior, adding new cells to the schedule.

5.5.2. Schedule Reset

Each time a clear message is sent from the sender to the receiver, it results in resetting the schedule on both sides. This means that all allocated cells so far should be removed, and the schedule adaptation should be initiated again for adding cells. By default, the sf-simple implementation does not include a mechanism to remove a schedule upon sending or receiving a CLEAR message. The handling of CLEAR messages is specifically defined in the *sixp.c* source file within its *sixp_input* and *sixp_output* functions, which are very simple handling routines.

In the *sixp_input* function, the only action taken when a CLEAR request is received is to reset the next sequence number for the corresponding neighbor. Additionally, the code logs an informational message to indicate that the neighbor's next sequence number has been reset. Similarly, in *sixp_output*, also same Upon success, it resets the neighbor's next sequence number, with a corresponding log message. To solve the lack of a removing mechanism, we defined a function to remove all the links from the schedule, which is called '*remove_all_links ()*'. This function is designed to iterate through the links in a given slotframe and remove each link individually. In this function, we used '*tsch_schedule_remove_slotframe ()*' defined in the '*tsch-schedule.c*' module to remove an entire slotframe along with all its links. However, the problem with using this function alone is that it removes all cells, including the "minimal cell". When the minimal cell is removed, both the sender and receiver lose their fundamental shared cell providing minimal connectivity, This results in the inability for the sender and receiver to, after removing their allocated cells, rebuild their schedule. The removal of the minimal cell disrupts the foundational connectivity necessary for essential communication, preventing the reconstruction of a basic schedule framework for further transmissions and transactions.

To address the problem of removing minimal cell, we used another function defined in '*tsch-schedule.c*' source called '*tsch_schedule_create_minimal()*'. This function, when called, creates a 6TiSCH minimal schedule that includes a single Tx|Rx|Shared slot using

the broadcast address. By doing so, after removing all cells, including the minimal cell, we use '*tsch_schedule_create_minimal()*' to recreate a minimal schedule sufficient for starting the adaptation of the schedule.

We placed both '*remove_all_links()*' and '*tsch_schedule_create_minimal()*' in both 'sixp_output' and '*sixp_input*' functions where CLEAR requests are handled. This ensures that, upon receiving or sending a CLEAR message, the schedule is appropriately cleared, and a minimal schedule is created to allow for further schedule adaptation. The snippet code illustrating this implementation is provided in Listing 5.4.

```
1  void sixp_input() {
2      // ...
3      if ((pkt.code.cmd == SIXP_PKT_CMD_CLEAR) && (nbr = sixp_nbr_find(
           src_addr)) != NULL) {
4          LOG_INFO("6P: sixp_input() reset nbr's next_seqno by CLEAR Request\n
               ");
5          sixp_nbr_reset_next_seqno(nbr);
6          remove_all_links();
7          tsch_schedule_create_minimal();
8      }
9      // ...
10 }
11
12 // -------------------------------------------------------------------
13
14 void sixp_output() {
15     // ...
16     if (code.cmd == SIXP_PKT_CMD_CLEAR) {
17         LOG_INFO("6P: sixp_output() reset nbr's next_seqno by CLEAR Request\
               n");
18         sixp_nbr_reset_next_seqno(nbr);
19         remove_all_links();
20         tsch_schedule_create_minimal();
21     }
22     // ...
23 }
```

Listing 5.4: Schdule Reset mechanism in sixp_input and sixp_output.

5.5.3. Adaptation decision point

As discussed earlier, for this experiment, our decision point for adding cells relies on tracking the number of used cells. In this section, we will explain the modifications we made to achieve this. In this experiment, due to some limitations caused by schedule resetting, we were unable to maintain the settings used in the main experiment related to measuring schedule adaptation duration. For example, we changed the sending UDP packet interval to periodic without using any sent packet threshold because our cell addition no longer depends on the UDP sending window. It is crucial to highlight that, due to this modifications, the durations collected for this experiment might be slightly different from the durations collected when the UDP window served as our cell addition

checkpoint. The primary objective of this experiment is to demonstrate how schedule inconsistency can induce changes in adaptation time. It is important to note that the intention is not to compare these results with those of the main experiment. Instead, the focus is on showcasing the impact of schedule irregularities on adaptation time,

To accurately track the utilization of cells, we used the *tx_count* counter, defined in the *tsch-slot-operation.c* source file. The *tx_count* variable serves as a reliable indicator of successful packet transmissions. It is incremented each time a packet is transmitted without encountering errors. However, since this counter is periodically incremented as the sender sends UDP packets periodically, we needed to define a threshold for determining when to consider adding a new cell.

```
1 if (tx_count % 3 == 0 && tx_count > 0) {
2     add_links_flag = 1; // Set the flag to indicate adding links
3 }
```

In the code snippet above, we check if the *tx_count* is a multiple of 3 and greater than 0. By requiring tx_count % 3 == 0, we establish a cyclic pattern where the cell addition process is activated at regular intervals, specifically every third successful transmission. If these conditions are met, we set the *add_links_flag* to 1. This flag serves as an indication to add new cells. By using this threshold, we introduce a mechanism to trigger cell addition based on the number of successful packet transmissions, providing a more adaptive approach to the changing network conditions.

5.6. sf-simple Challenges

In the course of our experiments, we encountered several challenges while working with the SF-Simple implementation, that required solutions. In this section, we explore these challenges and the possible solutions we devised to address them.

5.6.1. Lack of Bandwidth Monitoring and Adaptation Mechanisms

One of the initial challenges we encountered with SF-Simple was its inability to monitor the available bandwidth or adapt to traffic changes. The absence of bandwidth monitoring and adaptation mechanisms would hinder nodes from making informed decisions about adding or removing cells from their communication schedule. Due to time constraints, implementing these mechanisms as per the defined approach in rfc 9033 wasn't feasible within the scope of our study. As a result, we introduced our own approach, leveraging exchanged UDP packets between the sender and receiver. In our approach, we defined a sending window, representing the transmission of a predefined number of packets from the sender to the receiver. When this sending window is completed, it serves as an indicator similar to reaching the threshold of *MAX_NUM_CELL* in the realistic approach. We

assume that cell usage has exceeded its high threshold, prompting the necessity to add a new cell.

While our approach may not mirror the realistic approach precisely, it allows us to achieve the desired results for our experiments. In our study, the focus is on schedule adaptation, where cells are periodically added to the schedule until it reaches its maximum length. Given this emphasis on growing the schedule length, our bandwidth monitoring approach proves sufficient for the purposes of our experiments.This adaptation mechanism ensured the feasibility of our experiments within the given constraints and served as a suitable solution for the unique demands of our study.

5.6.2. Lack of Transaction Timeout Handler

sf-simple was missing a transaction timeout handler, which led to challenges with transaction failures. Without this mechanism, there was no built-in way to automatically handle timeouts for retransmission attempts. To address this challenge, we proactively implemented our custom transaction timeout handler within sf-simple. This handler played a crucial role in our experiments and simulations, ensuring that if a response was not received within a specified time frame, a timeout event would be triggered. In our experiments, we typically set timeout values to 10 seconds and 20 seconds, based on key performance indicators comparisons. The timeout mechanism itself is primarily implemented through a callback function aptly named 'timeout', Seeing the code snippet below:

```
1  static void timeout(sixp_pkt_cmd_t cmd, const linkaddr_t *peer_addr) {
2
3      sixp_nbr_t *nbr;
4      in_trans = false;
5      nbr = sixp_nbr_find(peer_addr);
6      if (init_seqno == 0) {
7          sixp_nbr_set_next_seqno(nbr, 0);
8      } else {
9      sixp_nbr_decrement_next_seqno(nbr);
10     }
11 }
```

This function is invoked when the system determines that a transaction has exceeded the predefined time limit. To enhance the timeout handling process, we introduced the *'in_trans'* flag, which is set to false within the 'timeout' function. This signifies the end of an active transaction, preventing the initiation of new transactions while the system is still processing a previous one. Originally, in the absence of the *'in_trans'* flag, the sf-simple module immediately started a new transaction with an incremented sequence number when a transaction failure occurred. However, this immediate initiation was not aligned with standard transaction handling practices. To address this, we introduced the *'in_trans'* flag, which serves as a safeguard to ensure that the system is aware of the occurrence of a transaction failure before initiating a new one. By tracking the *'in_trans'*

flag, sf-simple now adheres to a more standardized and controlled approach, allowing the 'timeout' function to work as intended. Additionally, as part of our implementation, we defined a decrement action in the 'timeout' function, which is covered in more detail as the second challenge in the next section.

5.6.3. Sequence Number Handling During Retransmissions

Another challenge we faced was associated with how sequence numbers were handled during retransmissions. In sf-simple, the sequence number was incremented for each retransmission attempt due to 6P transaction failures and there was no mechanism in place to handle this issue. This presented a challenge because, ideally, retransmissions should have the same sequence number as the original transaction that failed. To mitigate this challenge, we introduced a sequence number management mechanism. Initially, assigning an incremented sequence number to each retransmission posed issues, particularly when dealing with high failure probabilities (greater than 0.6). In such cases, the program could enter an infinite execution loop due to repeated increments of the sequence number. To resolve this, we implemented a sequence number "decrease" function. In the event of a retransmission, the sequence number would decrease to its previous value instead of incrementing. This adjustment ensured that the system would not enter an infinite loop and allowed for smoother operation in the face of high failure probabilities.

5.6.4. Missing 6P Error and Schedule Inconsistency Handler

According to RFC 9033 [23], a minimal scheduling function like SF-Simple should be capable of handling 6P errors and schedule inconsistencies. However, SF-Simple did not provide any built-in error or inconsistency handling mechanisms. In the '*sf_simple_driver*' structure, the 'error' field is set to 'NULL', indicating that there is no dedicated internal error handler associated with SF-Simple. When a schedule inconsistency is detected in the '*sixp.c*' source file, the '*handle_schedule_inconsistency*' function is called. This function checks if an error handler is available in the driver by examining the '*sf->error*' field ('*sf->error != NULL*'). In the case of SF-Simple, since the 'error' field is set to 'NULL', no specific action is taken for the internal error related to schedule inconsistency. As a result, SF-Simple lacks a mechanism to address and resolve scheduling inconsistencies.

Although SF-Simple has a limitation in handling certain 6P errors and schedule inconsistencies due to the absence of an internal error handler, this limitation did not significantly impact the results of our study. Specifically, errors such as 6P invalid transactions were not particularly relevant to our study objectives, and we did not encounter such errors. In fact, the only error that held significance for our research was schedule inconsistency, Which we explained in the previous section by detailing how we implemented clearing behavior to address this limitation.

6. Results and Disscusion

In the previous chapters, we discussed the essential aspects of our research. We focused on the Key Performance Indicators that serve as the foundation for evaluating the impact of 6P response failures on network performance. We detailed our experiment setups and the methods employed in gathering data. In this chapter, we move on to the next phase of our investigation. Our focus shifts to the analysis of the results and the implications of 6P response loss on network performance. This analysis represents the core of our research, where we aim to explore how 6P response losses have influenced our selected Key Performance Indicators.

In our experiments, we took into account a range of variable parameters that had a substantial impact on the results, which have been thoroughly outlined in Table 6.1, along with their corresponding values.

Parameter	Variable	Value
TSCH Slotframe length		101
TSCH Timeslot Duration		10 ms
6P Response loss Probability	P	{0.1 - 0.9}
Maximum number of cell	N	{5, 10, 15, 20}
Arrival sending rate	λ	N - 1
Timeout Value	T_{timeout}	{10s, 20s}

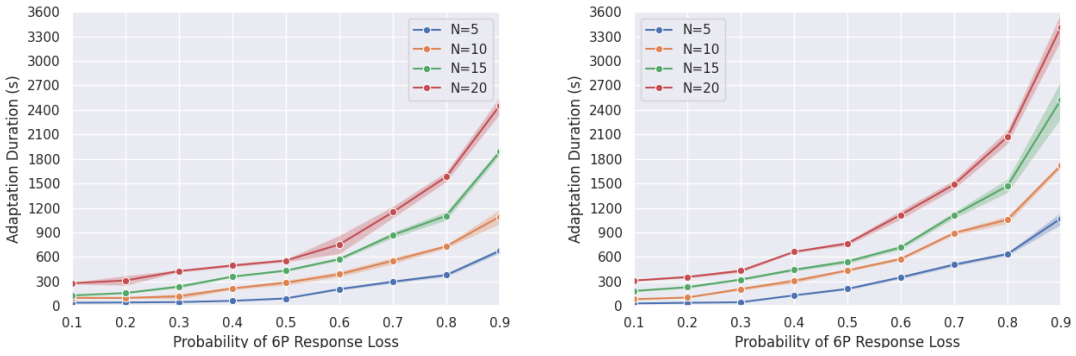
Table 6.1.: TSCH configuration and experiment parameters.

6.1. Schedule Adaptation Results

In this section, we will present results gathered from our experiments on schedule adaptation duration. The goal of this experiment was to observe how unsuccessful 6p transactions with different response loss probabilities affect the time it takes a node to adapt its schedule based on its traffic demands. We aimed to investigate how factors such as schedule length or timeout value can influence this adaptation time. The outcomes of our investigation are depicted in graphs 6.1a and 6.1b, providing a visual representation of the schedule adaptation durations.

Our experimental findings are presented with a 95% confidence interval in two distinct graphs that illustrate schedule adaptation duration. These graphs correspond to two different experiment timeout settings: one with a 10-second timeout and the other with a 20-second timeout. The schedule adaptation durations are precisely measured for each response loss probability and across four schedule lengths (N = 5, 10, 15, 20). In these graphical representations, the y-axis represents schedule adaptation time in seconds, the x-axis covers the range of response loss probabilities from 0.1 to 0.9, and each line on the graph depicts the trend of schedule adaptation time for a specific schedule length.

First, we start with the graph represented with a timeout value of 10 seconds. As a general observation, it can be seen in this graph that the time duration for each schedule length exhibits an increasing trend as the response loss probability increases. We can infer that all schedule lengths show the same increasing pattern. For a loss probability of 0.1, where the failure of 6p transactions is negligible, the time durations for all schedule lengths are in a closer range. As the loss probability grows, this difference also becomes more pronounced. For example, we can observe that the adaptation time at a probability of 0.1 for schedule lengths N = 5, 10, 15 falls within the range of less than 150 seconds with minimal differences. However, in the extreme case where the loss probability is 0.9, this difference becomes significantly noticeable. The consistent growth in adaptation time for all schedule lengths is apparent, but notably, for schedule length N = 20, this growth becomes steeper after a loss probability of 0.6. Furthermore, there is a sharp increase in adaptation time after a loss probability of 0.8, which is prominently observed for all schedule lengths. For instance, in the case of schedule length N = 20, it escalates from 1500 seconds to over 2400 seconds, roughly equivalent to 40 minutes.



(a) Adaptation time with Timeout 10s

(b) Adaptation time with Timeout 20s

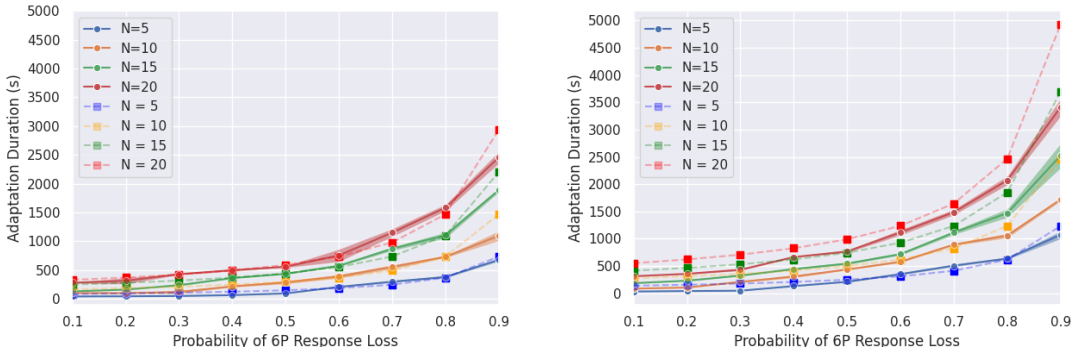
Figure 6.1.: Schedule Adaptation Duration.

If we examine the graph 6.1b with a timeout value of 20 seconds, we observe the exact same increasing pattern for each schedule length across different loss probabilities. However, there is a notable difference: the adaptation time values, compared to the results with a 10-second timeout, are larger, which is expected due to the longer timeout duration. Similar to the 10-second timeout graph, in this graph, we can observe that at higher loss

probabilities, adaptation times exhibit a larger difference range for each schedule length. In this timeout setting, following a probability of 0.9, all schedule lengths experience a very sharp and steep increase. For larger schedule lengths, specifically N=20, this increase exceeds 3300 seconds, approximately equivalent to 55 minutes, marking a significant value. This trend is also notable for a schedule length of N=5, where it takes approximately more than 1000 seconds for schedule adaptation.

6.1.1. Analitycal Model measurments

In Chapter 4, we introduced an analytical model (Equation 4.2), which served as a predictive tool for estimating the duration of schedule adaptation. This model took into account varying 6P response loss probabilities and timeout values as critical factors affecting the schedule adaptation process. Now In this section, our aim is to validate the accuracy of our analytical model by comparing the results obtained from its mathematical measurement with the empirical results of our experiments for schedule adaptation, presented in Figures 6.1a and 6.1b. The results from mathematical measurements are achieved for two timeout values, which match our empirical experiments (10s and 20s). These results are depicted in Figure 6.2a for a timeout value of 10s and in Figure 6.2b for a timeout of 20s. The mathematical results are represented by dashed linear trends with square markers, while the results achieved from our experiments are shown as continuous lines with confidence interval ranges.



(a) Results for Timeout 10s

(b) Results for Timeout 20s

Figure 6.2.: Analytical vs Experimental results.

For the 10-second timeout value, as depicted in Figure 6.2a, we can observe minimal differences between the results obtained from the analytical model and empirical experiments. In most probability values, we find a close match between the values. This observation is more pronounced for probabilities less than 0.8. However, for a probability of 0.9, our analytical model appears to overestimate the schedule adaptation duration, as indicated by the results. For example, for a schedule length of N=20, this duration is estimated at

approximately 3000 seconds, roughly 50 minutes, while we measured a shorter duration in the empirical experiment, approximately 2500 seconds (41 Minutes).

Figure 6.2b displays the results for a 20-second timeout value. In this figure, overestimation can be observed for schedule lengths N=15 and N=20. However, for shorter schedule lengths, it is evident that the empirical results align more closely with the estimated values. Similar to the trend seen in the results for the 10-second timeout, overestimation is particularly noticeable for a probability of 0.9. Notably, the overestimated values for the 20-second timeout are significantly higher than those for the 10-second timeout.

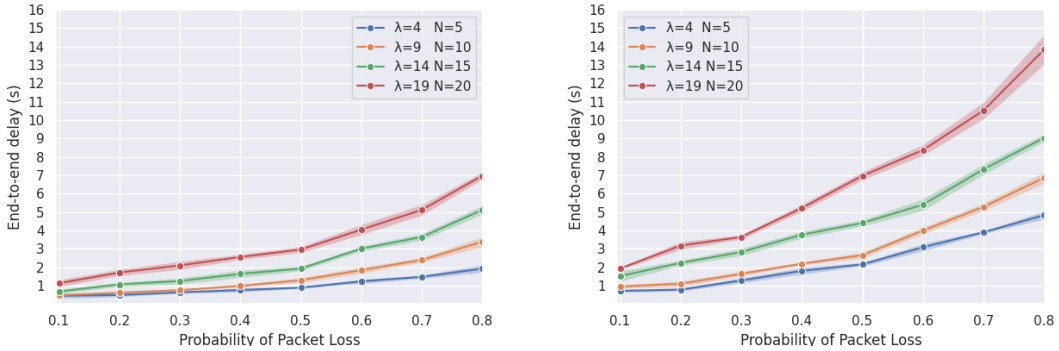
For instance, if we focus on a schedule length of N=20, the estimated duration is approximately 1500 seconds higher than the empirical results. This trend is consistent for a schedule length of N=15 as well. However, it's worth mentioning that for a schedule length of N=5, the estimated values in both cases are closer to the empirical values compared to other schedule lengths. While there may not be exact matches at every point, what's clear is that the growing pattern remains consistent for both cases. This indicates that an increase in the probability of 6P loss directly correlates with an extension of the time duration required for schedule adaptation.

6.2. End-to-end Delay Results

In our exploration of the impacts of 6P transaction failure, we delved into the second key performance indicator: End-to-End Delay. Through a series of experiments, we measured the end-to-end delay under the same parameters as our previous KPI, schedule adaptation duration. These parameters included timeout settings of 10 seconds and 20 seconds, schedule lengths of N=5,10,15,20, and response loss probabilities ranging from 0.1 to 0.9. The unique aspect of this experiment, as detailed in the methodology chapter, is the variation in the sending rate. In this particular experiment, the sender generated λ = N - 1 packets per slotframe. It's crucial to emphasize that these results were collected during the schedule adaptation phase, allowing us to observe how interruptions to schedule adaptation due to unsuccessful 6P transactions can impact the time needed for a packet to travel from sender to receiver, known as End-to-End Delay. Our results are presented in graphs 6.3a and 6.3b, corresponding to timeout values of 10 seconds and 20 seconds, respectively. These results are depicted with a 95% confidence interval, where the y-axis represents the amount of End-to-End Delay in seconds, and the x-axis denotes response loss probabilities. It should be noted that, due to the time-consuming procedure of schedule adaptation, we limited our data collection to a loss probability of 0.8, omitting 0.9. Since both graphs exhibit the same trend pattern, we anticipate that for a probability of 0.9, we would observe a similar pattern but at a higher level.

For a timeout value of 10 seconds, the End-to-End Delay for all schedule lengths, in the range of low loss probabilities, starts from less than 1 second and with a smooth increase, it reaches its maximum value at a loss probability of 0.8. For example, the

largest schedule length (N=20) with a sending rate of $\lambda = 19$ experiences an approximate delay of 7 seconds. Notably, as the loss probability increases, the difference in End-to-End Delay between schedule lengths becomes more pronounced.



(a) End-to-End Delay with Timeout 10s

(b) End-to-End Delay with Timeout 20s

Figure 6.3.: End-to-End Delay with Timeout 10s.

Graph 6.3b, represents results gathered for a timeout value of 20 seconds, although it follows the same increasing trend as graph 1.4, this increase is more pronounced. Particularly for the schedule length of N=20 with $\lambda = 19$, it starts to increase sharply after a loss probability of 0.3, reaching its maximum End-to-End value at a loss probability of 0.8, which is approximately 14 seconds, double the value at the same point for a timeout value of 10 seconds. Comparing values between the two graphs reveals that the End-to-End Delay for a timeout value of 20 seconds is significantly higher and has a more substantial portion of growth.

6.3. PDR Results

Packet Delivery Ratio (PDR) is an important metric used to evaluate the impact of 6P response failure in our study. PDR measures the proportion of successfully delivered data packets relative to the total number of packets transmitted within the network. Analyzing PDR measurements helps us understand the network's reliability when 6P transaction failures occur. In our experiment, we conducted PDR measurements in a specific scenario with a timeout value of 10 seconds during schedule adaptation, a schedule length of N = 10, and a sending rate of $\lambda = 9$ packets per slotframe. We used the default transmit Ring buffer defined by Contiki-NG, with a length of 8. To gather PDR values, we repeated the experiments five times to collect samples because we observed that the samples were closely clustered with minimal variability. We used the mean values of these samples to generate the plotted graph. The results of PDR associated with each response loss probability are depicted in Figure 6.4.

As evident from graph 6.4, we have presented results for the PDR using a bar chart. The x-axis represents the measured PDR during the scheduling process in percentage, while the y-axis shows the response loss probability. Our findings reveal that the PDR for a loss probability of 0.1 stands at approximately 82%, and with an increase in probability, PDR decreases. The decline from a probability of 0.2 to 0.5 follows a smooth trend but remains within the range of 70%-80%. From a probability of 0.6, the decrease becomes more pronounced, resulting in a PDR lower than 70%. This decline is especially extreme for a probability of 0.9, where a significant number of packets face drops, leading to a PDR of less than 50%. Viewing the graph as a general trend, it is evident that the overall trend is downward with respect to the increase in loss probability.

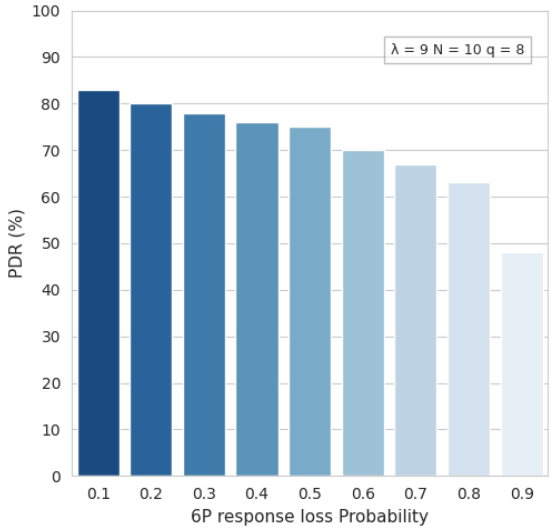


Figure 6.4.: Packet Delivery Ratio.

6.3.1. PDR Analitycal Model

Transition time $T_{\mu_1 \to \mu_M}$:

In Chapter 4, we introduced an analytical model designed as a tool for analyzing the Packet Delivery Ratio. Within this analytical model, we derived Equation 4.4, which facilitates the measurement of the time spent for the actual service rate to reach the target service rate. Now, in this section, our focus is on analyzing this time duration across different loss probabilities. We aim to investigate, for each loss probability, the amount of time required for the service rate to transition from μ_1 to μ_M. By utilizing Equation 4.4, we measured the time duration for each loss probability in seconds. The trends observed in the results are illustrated in Figure 6.5, represented through a linear

graph. Our measurements were conducted for a maximum number of cells, N=10, with a packet arrival rate of $\lambda = 9$. As observed from the graph, the time required for adapting the service rate for a loss probability of 0.1 is approximately 25 seconds.

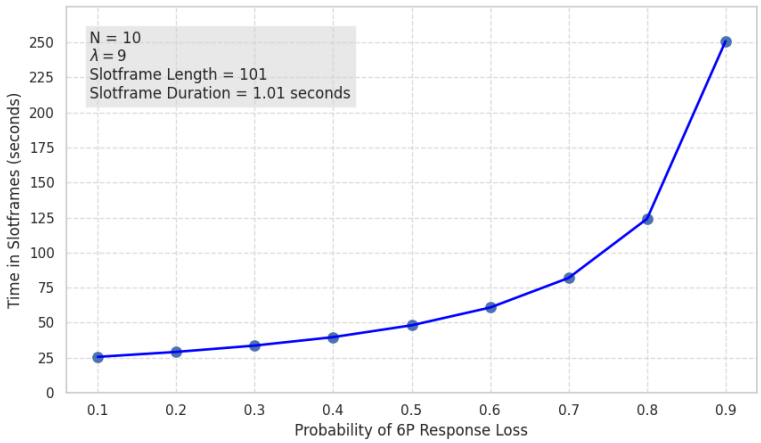


Figure 6.5.: Packet Delivery Ratio.

This time duration increases gradually until, for a loss probability of 0.5, it becomes double. This implies that if half of the 6P responses experience loss, we can expect double the time required for service rate adaptation compared to conditions where response loss is almost negligible. This increasing trend continues until a probability of 0.8, where a very sharp jump is evident, increasing from 125 seconds to 250 seconds at a probability 0.9. At this point, the time duration is 10 times more than the probability of 0.1, which proves that frequent instances of 6P transaction failures can noticeably increase the time duration for service rate adaptation. This, in turn, can result in negative effects on the performance of the network.

PDR Across Varied Queue Sizes :

The size of the Tx queue plays a significant role in determining the Packet Delivery Ratio of a network. The queue serves as a buffer where packets are enqueued before transmission, waiting for their turn to be sent. Hence, it is essential to investigate how different queue sizes impact PDR. In this section, we will analyze the effect of queue capacity (q) on PDR. Using equations 4.7 and 4.8, we measured PDR for four different queue sizes: q = 8, 16, 32, 64. The results of these measurements are presented in Figure 6.6, covering a range of response loss probabilities from 0.1 to 0.9. The reason we selected the queue (q) sizes of 8, 16, 32, 64 in our measurements is primarily due to the specific technical constraints of the Contiki-NG framework. Contiki-NG utilizes a ring buffer to store dequeued outgoing packets, which is essentially an array of pointers. It's important

to note that this ring buffer must have a power of two in size to enable atomic ring buffer operations. This technical requirement influenced our choice of queue sizes. We needed to conform to this constraint to ensure that our measurements aligned with the technical requirements of the Contiki-NG framework. For queue size 8, our observations reveal that

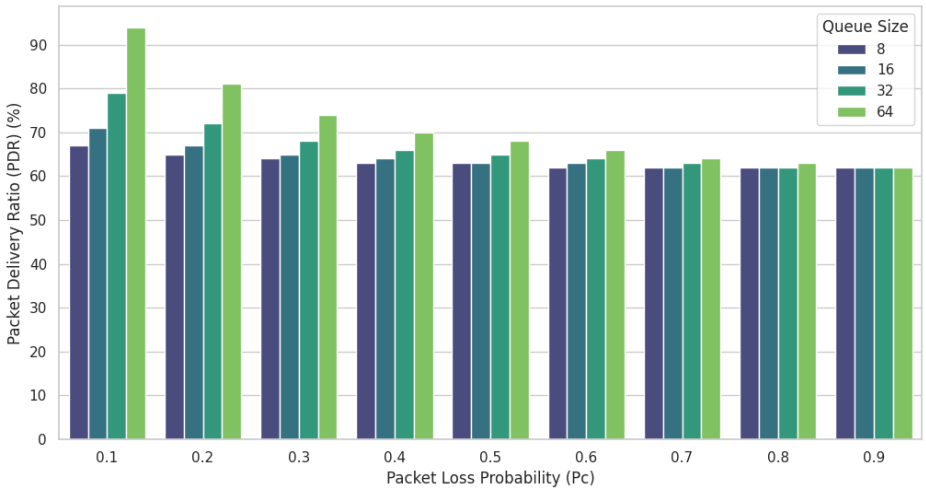


Figure 6.6.: PDR Variation across different Queue sizes.

as the probability of response loss increases, the packet delivery ratio gradually decreases. However, this decrease in PDR shows only minor differences for each probability value. In other words, the declining trend in PDR does not exhibit substantial variations for each probability value. For instance, when we compare a response loss probability of 0.1, which yields a PDR of 68%, to a response loss probability of 0.9, resulting in a PDR of 61%, we see that all PDR values fall within a relatively narrow range between 70% and 60%. Similar to queue size 8, the PDR trend for queue size 16 also exhibits a decrease as the response loss probability increases. However, when comparing it to the PDR trend for queue size 8, there is a slight improvement in PDR for probabilities in the range of 0.1 to 0.4. For instance, the PDR at a probability of 0.1 has improved and exceeded 70%. In contrast, for other probabilities, there isn't a significant difference when compared to queue size 8.

Increasing the queue size from 16 to 32 results in another significant improvement in PDR. This improvement is particularly notable at a response loss probability of 0.1, where the PDR reaches approximately 79%. Similarly, at a probability of 0.2, the PDR rises above 70%. However, it is important to note that despite these improvements, the general trend remains to decrease PDR as the response loss probability increases. This pattern is consistent with previous observations. For probabilities greater than 0.7, similar to other previous queue sizes, we observe almost no significant change in PDR. The overall decreasing pattern of PDR persists with a higher queue size of 64. Notably, for this queue

size, there is a significant increase in PDR, particularly for response loss probabilities ranging from 0.1 to 0.4. However, it is worth noting that even with this considerable improvement, probabilities greater than 0.5 still result in PDR values less than 70%. Interestingly, at a response loss probability of 0.9, the PDR remains relatively unchanged across all queue sizes. Another significant observation is the larger variance in PDR increase for lower probabilities. For instance, at a response loss probability of 0.1, we observe a PDR improvement of over 15% compared to queue size 32, reaching above 90%. However, for higher probabilities like 0.5, this improvement is approximately 4% in PDR.

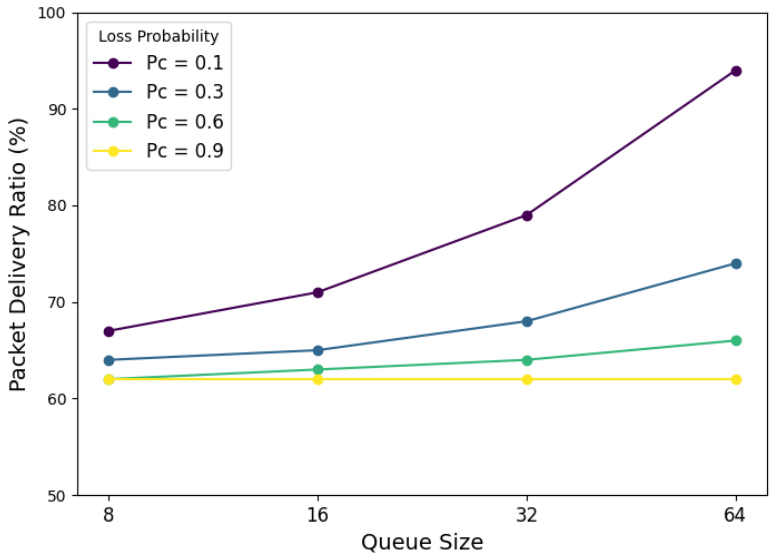


Figure 6.7.: The trend of PDR change across Queue sizes for selective probabilities.

To illustrate the trend of PDR changes across various queue sizes from the perspective of probabilities, we have presented an additional graph in Figure 6.7. The purpose of plotting this graph is to showcase the improvement in PDR for specific probabilities across different queues. In this graph, the x-axis represents the four queue capacities used in our measurements, and the y-axis displays the PDR value as a percentage. The trends are depicted for four selected probabilities: 0.1, 0.3, 0.6, and 0.9, serving as examples.

As shown in this graph, the PDR for probability 0.9 remains unchanged with an increase in queue capacity. For a probability of 0.6, the improvement by extending the queue size exhibits a slow growth. As the probability of loss decreases, this improvement becomes more significant and observable. Notably, for a probability of 0.1, we observe a substantial PDR improvement with a change in queue size. This graph proves that lower loss probabilities are more sensitive to queue size changes, highlighting a more pronounced impact as the queue size varies. Conversely, this sensitivity is inversely proportional for

higher loss probabilities, where changes in queue size have a comparatively less significant impact.

6.4. Energy consumption Results

In Wireless Sensor Network applications, energy efficiency is a crucial factor that aligns with one of its primary characteristics. Failures in 6P transactions can significantly impact this metric since retries and delays caused by unsuccessful transactions can result in increased energy usage. In this section, we aim to analyze the results we obtained from our experiments to demonstrate how unsuccessful 6P transactions can affect energy consumption. As mentioned earlier, in our experiment, we assessed the energy consumption of both the CPU and transmitter for the sender. This evaluation was carried out during schedule adaptation with a schedule length of N = 10. We collected our data through 5 repetitions, and the results are graphically represented in Figures 6.8 and 6.9. On these graphs, the Y-axis represents the energy consumption measured in millijoules, while the X-axis shows the range of 6P response loss probabilities, varying from 0.1 to 0.9. These graphs allow us to visualize the trends and changes in energy consumption as they relate to different 6P response probabilities.

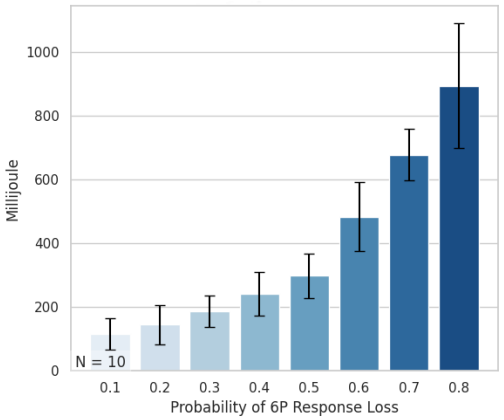


Figure 6.8.: Transmitter Energy Consumption.

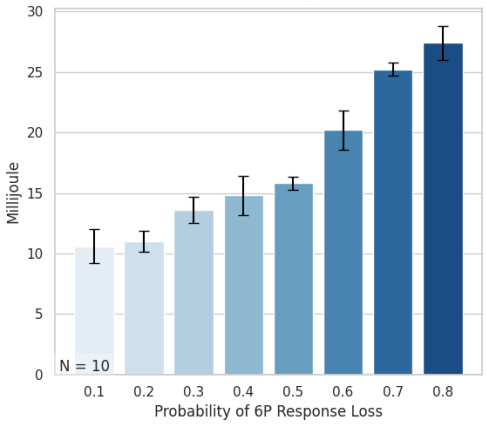


Figure 6.9.: CPU Energy Consumption.

To analyze our results, let's start by examining the energy consumption of the transmitter. As illustrated in Figure 6.8, energy consumption shows a clear upward trend across loss probabilities. This trend indicates that as the probability of 6P response loss increases, energy consumption also grows significantly. The upward trend initiates at a loss probability of 0.1, where the corresponding energy consumption remains below 200 millijoules. Transmitter energy consumption demonstrates a gradual increase until probability 0.5. Beyond this point, we observe that energy consumption continues to

rise, but at a more substantial rate. Continuing this progression, it becomes evident that in the most extreme case, with a probability of 0.8, energy consumption exceeds 900 millijoules.

Turning our attention to the graph depicting CPU energy consumption (Graph 6.9), we observe a similar increasing trend as with transmitter energy consumption. However, the values for CPU energy consumption are notably smaller. At probabilities 0.1 and 0.2, we can see that energy consumption stands in close range, approximately 10 and 11 millijoules, respectively. For other probabilities, we observe a more noticeable increase, particularly for probabilities exceeding 0.5, where this increase experiences a larger growth. For instance, it climbs from 16 millijoules to more than 25 millijoules at a response loss probability of 0.8.

6.5. Schedule Inconsistency Results

In this section, we present the results obtained from the experiments conducted to evaluate the impact of schedule inconsistency on adaptation time. In the discussion part, we delve into the interpretation of these results, exploring the implications gained from the observed outcomes.

Due to the specific nature of this experiment, we were limited in terms of parameters. As depicted in Figure 6.10, for this experiment, we considered a schedule length of $N = \{5, 10\}$ and a maximum clear probability of 0.5. The reason for this limitation is the disruption in the continuous flow of shaping the schedule. When nodes reset their schedule recently and, after adding some cells, receive a clear message again, they have to repeat this procedure. This cyclic resetting makes it difficult for the schedule to be shaped continuously and reach its defined maximum length, hindering our ability to collect its duration time. Due to this limitation, we relied on empirical trials, continuing our experiments by incrementing the values of our parameters to observe the maximum range of possibilities applicable for collecting the results. Based on our experience, we could only measure adaptation durations for these limited parameters. In addition to the clear probability range from 0.1 to 0.5, we also measured the adaptation time for the normal scenario where a node adapts its schedule without a reset. This served as a benchmark to have a meaningful comparison. However, it should be mentioned that at low loss probabilities, we didn't observe any resets.

As seen from the results, the normal schedule adaptation time for both schedule lengths of 5 and 10 is less than one minute. However, for a schedule length of 10, this time is double because its length is double. If we compare the normal duration with the duration across clear probabilities of 0.1 and 0.2, there is not much significant difference. The interesting observation is at a probability of 0.3, where a schedule length of 10 experiences a substantial increase in time (3 times more than normal duration), but the schedule length of 5 experiences a negligible increase. As the likelihood of schedule

inconsistency increases, we can see growth in adaptation time. This growth is very significant for probabilities of 0.4 and 0.5. Adaptation time for a schedule length of 5 reaches approximately 180 seconds to be built, and with a clear probability of 0.5, it exceeds 300 seconds. At these probability points, changes for a schedule length of 10 are significant. The extended time for schedule adaptation is so high that we were unable to collect any data (identified with N/A) because the node never be able to reach its maximum schedule length. For this case where we couldn't collect any adaptation time, we provided other results, which are presented in Table 6.2. We thought that these results could be useful as an alternative representation of impacts and could give better insight into the effects of schedule inconsistency.

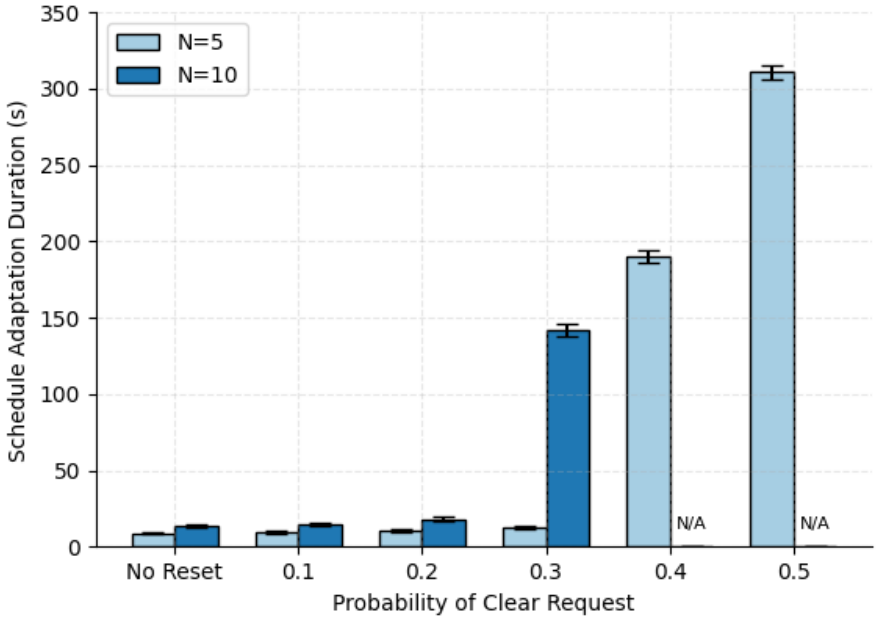


Figure 6.10.: Impact of schedule inconsistency on adaptation time.

These values are collected for both schedule lengths and cover just the probability range of 0.4 - 0.6. The Reset attempt parameter shows the average number of times nodes reset their schedule under the corresponding clear probability, and tx-count represents the final value of the transmission counter after the experiment is done or when we decided to stop the experiment. As seen in the table, the average reset attempts for a probability of 0.4 for a schedule length of 5 is 11.5 attempts, while this value for a schedule length of 10 is 230, indicating a significant difference. The corresponding differences in their tx counts are also substantial (127.8 vs. 2044).

Comparing values of this probability (0.4) with 0.6 for a schedule length of 5, we observe 30 times more reset attempts (315 attempts). In the Reset probability of 0.6 for a

N	Probability	Reset attempt	tx-count
5	0.4	11.5	127.8
	0.5	20.25	192
	0.6	315	1815
10	0.4	230	2044
	0.5	200	1475
	0.6	N/A	N/A

Table 6.2.: Number of Reset attempt and Tx-Count value.

schedule length of 10, results are not available because the duration time is very large, and the likelihood of reaching the maximum length is very low; thus, we didn't repeat our experiment for this case. For a schedule length of 10, we were not able to collect any time duration from a probability of 0.4. The reason for higher values in the probability of 0.4 compared to 0.5 is that we let the experiment for 0.4 probability run for a longer time, hoping to eventually obtain results. After being sure that no results could be collected for this probability, we ran the experiment for 0.5 for a shorter duration.

6.6. Discussion

6.6.1. Schedule Adaptation Analysis

In the previous section, we presented the results regarding schedule adaptation duration. Through graphical representation, we observed how adaptation duration changes across response loss probabilities. In this section, we will discuss further analysis and interpret these results to gain a deeper understanding of the effects that this metric can have on the performance of the 6TiSCH network. Regardless of the role of any parameter, in general, the provided results clearly indicate a direct relationship between response loss probability and the time required for schedule adaptation. This is obvious because, with a higher probability of transaction failure, the need for retrying the transmission of requests becomes more frequent. Consequently, the timeout occurs more often, disrupting the normal schedule adaptation procedure and leading to increased adaptation duration. Another significant observation regarding loss probabilities is the notable trend observed at the highest probability, 0.9. In both graphs, it is evident that, despite the gradually increasing pattern observed for probabilities ranging from 0.1 to 0.8, the adaptation time for a loss probability of 0.9 stands out significantly higher, exhibiting a substantial difference compared to the other probabilities. This is attributed to the fact that 90% of add requests are facing response loss at this probability, introducing considerable disruptions to the scheduling process and leading to an extended adaptation time.

Our findings reveal that, for cases with low response loss probabilities, minimal variation is observed between adaptation time across all schedule lengths. For example, in our experiments, no instances of transaction failure were observed when the loss probability was set to 0.1 and 0.2, except in rare cases, particularly when the schedule length was large, as seen in the case of N=20. The only variation in adaptation time within these probability ranges is associated with the differences in schedule lengths, where larger schedule lengths naturally require more time to adapt. This variation becomes more significant by increasing the loss probability. These observations prove that schedule length, as a parameter, plays a significant role in exacerbating the extended duration of the scheduling process when 6P transaction failures occur. It means that for larger schedule lengths, the required number of cells that must be added to the schedule is higher. Consequently, more 6P Add requests must be triggered, increasing the likelihood of encountering 6P transaction failures compared to lower schedule lengths.

The role of the timeout value as the second parameter is clearly observable. By comparing results for each timeout setting, we can see that in cases where the timeout value is chosen as 20 seconds, the adaptation duration is much longer than in cases where the timeout value was set to 10 seconds. The research conducted by Righetti et al.[26], which we also covered in Chapter 3, emphasized that the appropriate timeout value should be chosen to optimize the success rate of 6p transactions avoiding a value that is too high, which could lead to a higher transaction rate, or too low, which increases the chance of sequence number inconsistencies. Their recommendation for a suitable range of timeout values was between 40s to 60s. If we assume that we would use this timeout range in our experiments, the results for a timeout value of 20 seconds would become 2 times more for 40 seconds and 3 times for 60 seconds timeout, which, for high loss probabilities, is very high. This indicates that the timeout values recommended in this research, despite optimizing the success rate, may not be optimal for scenarios with elevated response loss probabilities.

In addition to our experimental results, we also provided values obtained from our analytical model. These results indicate that, in most cases, they align with our experimental findings. However, in certain instances, we observed some overestimation, particularly noticeable for values at a probability of 0.9, especially for schedule lengths of N=15,20 in the 20-second timeout setting. Nevertheless, our results demonstrate that Equation 4.2 can serve as a reliable predictive tool for estimating schedule adaptation time while considering the probability of transaction 6p failure. Despite some overestimation in specific cases, the analytical model generally provides valuable insights and complements our experimental findings, reinforcing its utility in predicting schedule adaptation durations under varying conditions.

Schedule adaptation is carried out in response to changes in traffic load. In our study, we have focused on its incremental growth, where the sender gradually adds cells to the schedule to enable it to handle high-traffic demands. Based on our results, we can conclude that the loss of 6P responses leads to an extended time for schedule adaptation. This extended time delays the allocation of the required number of Tx cells. In such cases,

the node has to continue using its currently available cells in the schedule, which may have a high usage rate, potentially leading to congestion.

To address this issue, MSF, by default, allocates extra cells. However, in scenarios where traffic load is high, and the failure probability of 6p transactions is elevated, this solution may also face challenges. The significant impact of parameters such as timeout and schedule length on the scheduling process shouldn't be forgotten. These parameters play an important role in adaptation time, and this role becomes more significant when the node is in a condition where 6p transactions are facing failure with a high probability.

6.6.2. End-to-End Delay Analysis

The impact of 6P transaction failure on End-to-End Delay is similar to the observations discussed for schedule adaptation duration. For this metric, we can see that as the probability of 6P transactions increases, End-to-End Delay also increases. Schedule length and timeout value also exacerbate this metric; their impacts are clearly observable in high-loss probabilities. Packet arrival rate (λ) as the third parameter also plays an important role in the escalation of delays. As we concluded, the extended schedule adaptation time due to 6P transaction failure causes delays in allocating required TX cells to handle incoming packets. This results in a shortage of transmission cells in the schedule, which cannot accommodate the transmission of all incoming packets upon arrival, leading to packet queuing. As more packets queue up, they spend more time waiting to be transmitted, further worsening End-to-End Delay. The waiting time can increase with an increasing number of packet arrivals, as the higher influx of packets results in a larger number of packets being enqueued for transmission within each slot frame. Consequently, the waiting time for transmission becomes longer, resulting in increased delays in delivering packets on time. In applications where real-time communication and low latency are critical, these increased delays can have significant consequences. For instance, in industrial automation or healthcare systems, timely and reliable transmission of data is paramount. The extended End-to-End Delay introduced by 6P transaction failures may lead to disruptions in control signals or vital information transfer, negatively impacting the overall system performance. In scenarios where rapid responses are essential, such as in autonomous vehicles or mission-critical communications, the heightened delays caused by 6P transaction failures could compromise the effectiveness and reliability of the entire system.

6.6.3. PDR Analysis

From the bar chart provided in Figure 6.4, it becomes evident that failures in 6P transactions have a significant impact on the network's reliability. This effect is particularly noticeable in the decreasing Packet Delivery Ratio (PDR), indicating a reduced ability to successfully deliver data packets. As the probability of failure increases, we can

anticipate a higher number of packet drops at the application layer, leading to reduced performance.

An unsuccessful 6P transaction for an "add request" results in the delayed allocation of the necessary Tx cells for transmitting enqueued packets. This shortage of Tx cells in the TSCH schedule translates to a lower service rate, requiring more slotframes to be passed to transmit outgoing packets. Simultaneously, with each elapsed slotframe, the sender generates additional packets at a rate of λ, exacerbating the situation. This results in an increased accumulation of packets in the Tx queue. However, the limited size of the Tx queue means it quickly becomes congested, reaching its capacity faster and, consequently, dropping more packets.

An effective solution for improving low PDR is to implement a larger queue size, providing more room for maintaining packets. This enhanced capacity significantly reduces the likelihood of the queue reaching its limit, thereby improving the rate of packet drops. This consideration has been incorporated into our analytical model, where we demonstrated how the PDR value can vary with changes in queue sizes.

A notable observation derived from the results of our analytical model is that, for high loss probabilities, the Packet Delivery Ratio remains consistently low across various queue sizes, with only marginal improvements. This phenomenon is evident in Figure 6.7, where even for a loss probability of 0.9, PDR doesn't show any improvement across all queue sizes. This observation can be attributed to the fact that higher loss probabilities lead to a larger value of L, representing the total number of unserved packets in the last round of schedule adaptation. Since q remains a constant value in the calculation of the number of dropped packets, the difference between L and q variables (L - q) becomes more substantial when L is larger compared to smaller L values. In simpler terms, higher loss probabilities lead to the generation of more packets and, consequently, more packet loss. The queue sizes (q) used in our measurements are not large enough to significantly improve PDR for the volume of generated packets and dropped packets associated with high loss probabilities, especially a loss probability of 0.9.

Our analysis demonstrates that error-free operation of 6P transactions plays an important role in ensuring the reliability of applications where successful packet delivery is critical. Any occurrence of failure in 6P transactions, particularly in scenarios with high probabilities of failure, undermines the integrity and effectiveness of these applications. Additionally, systems with limited memory facilities may face even greater challenges in maintaining the reliability of packet delivery under such circumstances. This highlights the need for robust 6P transactions in these critical application domains.

6.6.4. Energy Consumption Analysis

Based on our results, we can conclude that the energy consumption of sensor nodes is directly impacted by unsuccessful 6P transactions in an increasing manner. As each 6P transaction experiences failure, the need for retransmission arises, causing the transmitter

to become active more frequently than in an ideal situation where no failures in 6P transactions occur. On the other hand, during timeouts, the sender remains in a receiving (RX) state, waiting for a response. When viewed from the CPU's perspective, extending the time for schedule adaptations necessitates additional CPU operations, such as synchronization and data processing. These additional operations lead to increased energy consumption for both the transmitter and CPU. Another significant observation is the difference between the energy consumption values achieved for the CPU and transmitter. Specifically, CPU energy consumption is much lower than that of the transmitter. This suggests that, during schedule adaptation, the transmitter operates more actively than the CPU. Several factors contribute to this observation. Firstly, according to OpenMote-B's datasheet, the transmitter draws approximately 10 times more electric current than the CPU. Secondly, unsuccessful transactions necessitate transmission retries, leading to heightened transmitter activity.

Additionally, our experimental setup includes UDP traffic flows, which require the transmitter to be active before any cell addition occurs. These combined factors underscore the transmitter's increased activity and energy consumption in comparison to the CPU. An increase in energy consumption poses a significant challenge for battery-powered devices with limited energy reserves. The negative impact of elevated energy consumption can result in a shortened battery life, reducing the operational lifespan of these devices.

6.6.5. Schedule Inconsistency Analysis

Our results presented in the previous section clearly indicate that when the probability of schedule inconsistency increases, adaptation time is significantly influenced, resulting in an increase. This substantial impact is due to the removal of the entire communication schedule and the need to rebuild it. In this case, the node must invest time to reallocate cells that were removed before and allocate new cells, shaping the entire schedule length. If, during this procedure, schedule inconsistency is detected again, followed by the sending of a clear message, additional attempts are needed to repeat all steps. As the probability increases, this resetting occurs more frequently, worsening the adaptation time. This phenomenon can become even worse when the schedule has a long length because larger schedules naturally require more time to be built, providing more opportunities for schedule inconsistency to occur. This relationship is observable in our results, as we can see that for a schedule length of 5, we could continue our experiment for higher probabilities and collect more data compared to a schedule length of 10. The limited parameters of probability and schedule length prove that adaptation time is very sensitive to resetting the schedule, especially in conditions where the likelihood of schedule inconsistency is higher.

If we want to compare extended adaptation time due to the loss of 6P response with extended adaptation time due to schedule inconsistency, our results conclude that schedule inconsistency has much more negative effects. This is because a failure in a 6P transaction only involves a single communication cell, and in the absence of this cell, other cells

in the schedule can be utilized to handle an increased portion of traffic demand. This results in decreased quality and performance of the network. However, the extended time duration caused by resetting the schedule will involve all communication resources. This is because nodes remove all of their cells, disrupting the entire communication between nodes, and the network can be left in a more vulnerable state with a larger portion of communication resources unavailable during the adaptation process. This significantly impacts the network's ability to efficiently handle communication demands and can lead to more severe performance degradation compared to the loss of a single 6P transaction.

This conclusion is clearly evident from our results, where adaptation time, by increasing probabilities, indicates significant changes in the case of schedule inconsistency compared to cases where 6P transaction failure occurs. Graphs related to response loss probabilities show a steady increase in time, but in the case of schedule reset, this increase follows a huge growing trend, reaching a point where the collection of data becomes impossible in some cases. Looking from another perspective, there is a relationship between schedule inconsistency and 6P transaction failure. We can also conclude that an unsuccessful 6P transaction may lead to a situation where the node has to trigger a clear request, causing MSF to decide to take action in the form of applying clear behavior. From this perspective, this fact can be true that 6P transaction failures, which lead to schedule reset, are more detrimental to the entire performance of the network.

6.7. Summery

In this chapter, we conducted an in-depth analysis of how transaction failures in 6P affect the key performance indicators. Our research has provided significant insights into the relationship between 6P transaction failures and critical metrics, such as end-to-end delay, packet delivery ratio, energy consumption, and schedule adaptation duration. Our study has shown that 6P transaction failures have a cascading effect on the performance of the network. As the probability of transaction failure increases, the time required for schedule adaptation also increases. This leads to delays in the allocation of necessary transmission cells, amplifying end-to-end delays, causing a backlog of packets, and adversely affecting the packet delivery ratio. This also leads to an increase in the frequency of retransmissions and heightened activity of the transmitter, posing challenges for energy-constrained sensor nodes.

Additionally, we extended our analysis to schedule inconsistency, highlighting its substantial influence on adaptation time. As the probability of inconsistency increased, the need to rebuild the entire communication schedule became more frequent, especially for longer schedules. Comparatively, our study emphasized that schedule inconsistency had a more pronounced negative effect on adaptation time than 6P transaction failures.

7. Conclusion

7.1. Context

This chapter, as the concluding segment of our study provides a comprehensive overview of our key findings, a reflection on the research process, a discussion on the significance of our findings to the field, and a final wrap-up of our thesis.

Throughout this thesis, our main objective was to answer a fundamental question: How do failures in 6P transactions affect the stability and performance of 6TiSCH networks? To gain a comprehensive understanding, we explored further inquiries, such as the impact of schedule length, timeout value, sending rate, and the probabilities of transaction failures on the overall network performance. We also investigated the influence of schedule inconsistency, which is an essential aspect of our study.

To answer these questions, we conducted a series of experiments designed to deliberately induce disruptions in 6P transactions. We varied the parameters systematically to observe their effects on key performance indicators, such as schedule adaptation duration, end-to-end delay, packet delivery ratio, and energy consumption. Our multifaceted approach allowed us to closely examine the interplay between transaction failures and various network parameters.

The results obtained from our experiments revealed that the failure of 6P transactions can significantly degrade the performance of the network. The loss of 6P response messages, which leads to unsuccessful 6P transactions between neighboring nodes, substantially increases the time duration required for the sender node to adapt its TSCH schedule in response to increased traffic. This effect is especially noticeable when a node needs to construct a schedule that contains a large number of cells and when the probability of dropped 6P messages is high. Additionally, the choice of the timeout value significantly affects the duration, with larger values causing further extensions. Delay in schedule adaptation caused by 6P transaction failure not only results in a lower service rate but also leads to an increase in end-to-end delay, which is a critical metric for real-time applications. Moreover, high traffic loads can exacerbate the end-to-end delay, making it even more detrimental to the performance of the network.

Delayed resource allocation hampers the network's ability to efficiently allocate resources, resulting in a reduced service rate. This reduced service rate, particularly under high traffic loads, raises the risk of packet loss. Furthermore, the capacity of the Tx queue is pivotal in network performance. Low Tx queue capacities can lead to an increased risk of

packet loss, especially when the queue is overwhelmed by the volume of data, resulting in a higher risk of data loss. Our results also showed that the energy consumption of nodes in the network is directly influenced by how efficiently 6P transactions operate. Unsuccessful 6P transactions necessitate nodes to wait for responses more frequently, and in the absence of responses, they retry to transmit 6P messages, resulting in increased energy consumption.

Additionally our study reveals a significant correlation between the probability of schedule inconsistency and an extended adaptation time in communication networks. As the likelihood of inconsistencies rises, frequent schedule resets substantially extend the adaptation process, with a more pronounced impact in scenarios with higher probabilities and longer schedule lengths. Comparing the impact of extended adaptation time due to 6P transaction loss with the impact of schedule inconsistency, we find that schedule inconsistency has a more detrimental effect. While a failed 6P transaction affects a single cell, schedule resets disrupt the entire network, leading to a larger portion of communication resources being unavailable. This vulnerability during adaptation can result in more severe performance degradation.

The complex relationship between 6P transactions and performance metrics highlights the importance of having robust and efficient mechanisms in 6TiSCH networks. To overcome the challenges caused by transaction failures, it is essential to carefully consider parameters such as timeout values, schedule length, and queue sizes, and explore adaptive strategies and optimization techniques. These findings not only offer valuable insights to improve the reliability and efficiency of 6TiSCH networks but also provide a foundation for future research and potential mitigation strategies.

7.2. Study Limitations

This thesis, despite its contributions and insights, is not without its limitations. It is crucial to acknowledge these constraints.

As previously mentioned, the Scheduling function provided by Contiki-NG was not a complete version, lacking some features introduced in RFC 9033. Due to time constraints in our thesis, we made every effort to implement the identified shortcomings related to our study. However, addressing these limitations consumed a significant portion of our research efforts. If these limitations did not exist, we could get more precise results aligned with real-world scenarios, allowing us to potentially focus more on other factors and aspects discussed in the future work section. The primary shortcoming of this scheduling function was its absence of a traffic monitoring mechanism, which could, in some instances, impact the results. Despite the limitations of time, we made every effort to addressing this gap, with the aim of delivering a more realistic solution.

7.3. Future Work

Our study focused on exploring the effects of 6P transaction failures on 6TiSCH networks, particularly on the add request message. However, we acknowledged that there are different types of 6P messages and that each type has its own unique consequences. Due to time constraints, we only looked into the add request message. To broaden the scope of understanding, future research could investigate the impacts of 6P transaction failures on other message types within the 6TiSCH network. We recommend examining how the failure of different 6P message types affects network stability and performance, as this could provide more extensive insights into the consequences of 6P transaction failures. Furthermore, our study primarily focused on 2-step transactions, but future research should delve deeper into the dynamics of 3-step transactions to reveal additional intricacies and challenges posed by this extended transaction model. Another promising area for future research is to evaluate the effects of 6P transaction failures in a network with multiple hub nodes, which would add an additional layer of complexity to the study. Examining how 6P transaction failures propagate through such multi-hub topologies could uncover additional insights into the resilience and performance of 6TiSCH networks.

In addition to the minimal scheduling function, several alternative scheduling functions have been introduced by researchers, such as Orchestra and many others. As part of future work, an avenue worth exploring is the investigation of the impacts of 6P transaction failures using these alternative scheduling functions. Each scheduling function operates with its unique mechanisms and features in responding to 6P errors. Studying the effects of 6P transactions under different scheduling functions may yield varied results and outcomes, providing a detailed understanding of the network's behavior in the face of transaction failures. In summary, future investigations can expand upon our research by exploring the impact of 6P transaction failures across different message types, transaction models, and network topologies, as well as by incorporating the utilization of other scheduling functions. Addressing these aspects can lead to a more comprehensive understanding of the robustness and potential vulnerabilities inherent in 6TiSCH networks.

A. Acronyms

WSN Wireless Sensor Networks

IoT Internet of Things

TSCH Time-Slotted Channel Hopping

LR-WPANs Low-Rate Wireless Personal Area Networks

IEEE Institute of Electrical and Electronics Engineers

IPv6 Internet Protocol version 6

6TiSCH IPv6 over IEEE 802.15.4e TSCH

KPIs Key Performance Indicators

PHY Physical

MAC Medium Access Control

TDMA Time Division Multiple Access

FDMA Frequency Division Multiple Access

IPv6 Internet Protocol version 6

6TiSCH IPv6 over IEEE 802.15.4e Time-Slotted Channel Hopping

6top 6TiSCH Operation Sublayer

MSF Minimal Scheduling Function

SF Scheduling Function

6P 6top Protocol

RPL Routing Protocol for Low-Power and Lossy Networks

CoAP Constrained Application Protocol

EB Enhanced Beacon

AutoTxCell Autonomous Transmit Cell

AutoRxCell Autonomous Receive Cell

PDR Packet Delivery Ratio

List of Figures

List of Tables

Bibliography

[1] Senchun Chai, Zhaoyang Wang, Baihai Zhang et al. *Wireless Sensor Networks*. Springer Singapore, 2020. DOI: 10.1007/978-981-15-5757-6. URL: https://link.springer.com/book/10.1007%2F978-981-15-5757-6 (visited on 21st May 2021).

[2] Andreas Reinhardt, Sebastian Zöller and Delphine Christin. 'Wireless Sensor Networks and Their Applications: Where Do We Stand? And Where Do We Go?' In: *Proceedings of the 13th GI/ITG KuVS Fachgespräch „Drahtlose Sensornetze" (FGSN)*. 2014, pp. 1–4.

[3] J.A. Gutierrez, M. Naeve, E. Callaway et al. 'IEEE 802.15.4: a developing standard for low-power low-cost wireless personal area networks'. In: *IEEE Network* 15.5 (2001), pp. 12–19. DOI: 10.1109/65.953229.

[4] Ala' Khalifeh, Khalid A. Darabkh, Ahmad M. Khasawneh et al. 'Wireless Sensor Networks for Smart Cities: Network Design, Implementation and Performance Evaluation'. In: *Electronics* 10 (January 2021), p. 218. DOI: 10.3390/electronics10020218. URL: https://www.mdpi.com/2079-9292/10/2/218 (visited on 21st April 2021).

[5] Diego Dujovne, Thomas Watteyne, Xavier Vilajosana et al. '6TiSCH: deterministic IP-enabled industrial internet (of things)'. In: *IEEE Communications Magazine* 52.12 (2014), pp. 36–41. DOI: 10.1109/MCOM.2014.6979984.

[6] Uttara Gogate and Jagdish W. Bakal. 'Smart Healthcare Monitoring System based on Wireless Sensor Networks'. In: *2016 International Conference on Computing, Analytics and Security Trends (CAST)*. 2016, pp. 594–599. DOI: 10.1109/CAST.2016.7915037.

[7] Shuang-Hua Yang. *Wireless Sensor Networks Principles, Design and Applications*. London Springer, 2014.

[8] Chiara Buratti, Andrea Conti, Davide Dardari et al. 'An Overview on Wireless Sensor Networks Technology and Evolution'. In: *Sensors* 9 (August 2009), pp. 6869–6896. DOI: 10.3390/s90906869.

[9] Milica Pejanović Đurišić, Zhilbert Tafa, Goran Dimić et al. 'A survey of military applications of wireless sensor networks'. In: *2012 Mediterranean Conference on Embedded Computing (MECO)*. 2012, pp. 196–199.

[10] Luis Flores Luyo, Agostinho Agra, Rosa Figueiredo et al. *Vehicle routing problem for information collection in wireless networks*. HAL Archives Ouvertes, February 2019. URL: https://hal.archives-ouvertes.fr/hal-02176511 (visited on 28th November 2023).

[11] Tao Xu, Lina Gong, Wei Zhang et al. 'Application of wireless sensor network technology in logistics information system'. In: *Nucleation and Atmospheric Aerosols* (January 2017). DOI: 10.1063/1.4981549. (Visited on 28th November 2023).

[12] I. Howitt and J.A. Gutierrez. 'IEEE 802.15.4 low rate - wireless personal area network coexistence issues'. In: *2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.* Vol. 3. 2003, 1481–1486 vol.3. DOI: 10.1109/WCNC.2003.1200605.

[13] 'IEEE Standard for Low-Rate Wireless Networks'. In: *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)* (July 2020), pp. 1–800. DOI: 10.1109/IEEESTD.2020.9144691. URL: https://ieeexplore.ieee.org/document/9144691.

[14] Domenico De Guglielmo, Simone Brienza and Giuseppe Anastasi. 'IEEE 802.15.4e: A survey'. In: *Comput. Commun.* 88 (2016), pp. 1–24. URL: https://api.semanticscholar.org/CorpusID:38558199.

[15] Tadanori Matsui and Hiroaki Nishi. 'Time slotted channel hopping scheduling based on the energy consumption of wireless sensor networks'. In: *2018 IEEE 15th International Workshop on Advanced Motion Control (AMC)*. 2018, pp. 605–610. DOI: 10.1109/AMC.2019.8371162.

[16] Seungbeom Jeong, Jeongyeup Paek, Hyung-Sin Kim et al. 'TESLA: Traffic-Aware Elastic Slotframe Adjustment in TSCH Networks'. In: *IEEE Access* 7 (2019), pp. 130468–130483. DOI: 10.1109/access.2019.2940457.

[17] Mohamed Mohamadi and Mustapha Réda Senouci. 'Scheduling Algorithms for IEEE 802.15.4 TSCH Networks: A Survey'. In: *Lecture notes in networks and systems* (August 2018), pp. 4–13. DOI: 10.1007/978-3-319-98352-3_2. (Visited on 17th October 2023).

[18] Sarra Hammoudi, Saad Harous and Zibouda Aliouat. 'External Interference Free Channel Access Strategy Dedicated to TSCH'. In: *2018 IEEE International Conference on Electro/Information Technology (EIT)*. 2018, pp. 0350–0355. DOI: 10.1109/EIT.2018.8500259.

[19] Andreas Ramstad Urke, Øivind Kure and Knut Øvsthus. 'A Survey of 802.15.4 TSCH Schedulers for a Standardized Industrial Internet of Things'. In: *Sensors (Basel, Switzerland)* 22 (December 2021), p. 15. DOI: 10.3390/s22010015. URL: https://pubmed.ncbi.nlm.nih.gov/35009558/ (visited on 1st July 2022).

[20] Apostolos Karalis, Dimitrios Zorbas and Christos Douligeris. 'Collision-Free Advertisement Scheduling for IEEE 802.15.4-TSCH Networks'. In: *Sensors* 19 (April 2019), p. 1789. DOI: 10.3390/s19081789. (Visited on 24th October 2022).

[21] Xavier Vilajosana, Thomas Watteyne, Malisa Vucinic et al. '6TiSCH: Industrial Performance for IPv6 Internet-of-Things Networks'. In: *Proceedings of the IEEE* 107 (June 2019), pp. 1153–1165. DOI: 10.1109/jproc.2019.2906404. (Visited on 9th May 2021).

[22] Xavier Vilajosana, Thomas Watteyne, Tengfei Chang et al. 'IETF 6TiSCH: A Tutorial'. In: *IEEE Communications Surveys Tutorials* (2019), pp. 1–1. DOI: 10.1109/comst.2019.2939407. (Visited on 19th February 2020).

[23] Tengfei Chang, Mališa Vučinić, Xavier Vilajosana et al. *6TiSCH Minimal Scheduling Function (MSF)*. IETF, May 2021. URL: https://datatracker.ietf.org/doc/rfc9033/ (visited on 8th September 2023).

[24] David Hauweele, Remous-Aris Koutsiamanis, Bruno Quoitin et al. 'Thorough Performance Evaluation Analysis of the 6TiSCH Minimal Scheduling Function (MSF)'. In: *Journal of Signal Processing Systems (2022)* 94 (June 2021), pp. 3–25. DOI: 10.1007/s11265-021-01668-w. (Visited on 6th June 2023).

[25] Qin Wang, Xavier Vilajosana and Thomas Watteyne. *6TiSCH Operation Sublayer (6top) Protocol (6P)*. IETF, November 2018. URL: https://datatracker.ietf.org/doc/rfc8480/ (visited on 8th September 2023).

[26] Francesca Righetti, Carlo Vallati, Giuseppe Anastasi et al. 'Performance Evaluation the 6top Protocol and Analysis of its Interplay with Routing'. In: *IEEE* (May 2017). DOI: 10.1109/smartcomp.2017.7947029.

[27] Yasuyuki Tanaka, Pascale Minet, Thomas Watteyne et al. 'Accelerating 6TiSCH Network Formation'. In: *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. 2021, pp. 18–24. DOI: 10.1109/DCOSS52077.2021.00016.

[28] Francesca Righetti, Carlo Vallati, Giuseppe Anastasi et al. 'Analysis and Improvement of the On-The-Fly Bandwidth Reservation Algorithm for 6TiSCH'. In: *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. 2018, pp. 1–9. DOI: 10.1109/WoWMoM.2018.8449793.

[29] Maria Rita Palattella, Thomas Watteyne, Qin Wang et al. 'On-the-Fly Bandwidth Reservation for 6TiSCH Wireless Industrial Networks'. In: *IEEE Sensors Journal* 16 (January 2016), pp. 550–560. DOI: 10.1109/jsen.2015.2480886. (Visited on 30th October 2020).

[30] Tengfei Chang, Mališa Vučinić, Xavier V. Guillén et al. '6TiSCH minimal scheduling function: Performance evaluation'. In: *Internet Technology Letters* 3 (June 2020). DOI: 10.1002/itl2.170. (Visited on 12th March 2021).

[31] Karnish, Manas Khatua and Venkatesh Tamarapalli. 'IMSF: Improved Minimal Scheduling Function for Link Scheduling in 6TiSCH Networks'. In: *ICDCN* (January 2022). DOI: 10.1145/3491003.3491027. (Visited on 16th October 2023).

[32] David Hauweele, Remous-Aris Koutsiamanis, Bruno Quoitin et al. 'Pushing 6TiSCH Minimal Scheduling Function (MSF) to the Limits'. In: *HAL (Le Centre pour la Communication Scientifique Directe)* (July 2020). DOI: 10.1109/iscc50000.2020.9219692. (Visited on 16th October 2023).

[33] Esteban Municio, Glenn Daneels, Marijana Vučinić et al. 'Simulating 6TiSCH networks'. In: *Transactions on emerging telecommunications technologies* 30 (March 2019), e3494–e3494. DOI: 10.1002/ett.3494. (Visited on 23rd April 2023).

[34] Gihwan Kim, Sanghwa Chung and Minjae Kim. 'Delayed Cell Relocation in 6TiSCH Networks'. In: *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*. 2019, pp. 656–661. DOI: 10.1109/ICUFN.2019.8806109.

[35] Francesca Righetti, Carlo Vallati, Sajal K. Das et al. 'An Experimental Evaluation of the 6top Protocol for Industrial IoT Applications'. In: *2019 IEEE Symposium on Computers and Communications (ISCC)*. 2019, pp. 1–6. DOI: 10.1109/ISCC47284.2019.8969590.

[36] Marc Domingo-Prieto, Tengfei Chang, Xavier Vilajosana et al. 'Distributed PID-Based Scheduling for 6TiSCH Networks'. In: *IEEE Communications Letters* 20 (May 2016), pp. 1006–1009. DOI: 10.1109/lcomm.2016.2546880. (Visited on 14th October 2023).

[37] Yuvin Ha and Sang-Hwa Chung. 'Enhanced 6P Transaction Methods for Industrial 6TiSCH Wireless Networks'. In: *IEEE Access* 8 (2020), pp. 174115–174131. DOI: 10.1109/ACCESS.2020.3025943.

[38] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel et al. 'Orchestra'. In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems* (November 2015). DOI: 10.1145/2809695.2809714. (Visited on 25th March 2022).

[39] A. Dunkels, B. Gronvall and T. Voigt. 'Contiki - a lightweight and flexible operating system for tiny networked sensors'. In: *29th Annual IEEE International Conference on Local Computer Networks*. 2004, pp. 455–462. DOI: 10.1109/LCN.2004.38.

[40] Yasuyuki Tanaka, Ito T and Fumio TERAOKA. '6TiSCH Scheduling Function Design Suite founded on Contiki-NG'. In: *Journal of information processing* 30 (January 2022), pp. 669–678. DOI: 10.2197/ipsjjip.30.669. (Visited on 25th September 2023).