FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Synchronization and Routing for Cooperating Agents

Ana Morais

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Luis Almeida

October 31, 2024

© Ana Morais, 2024

Resumo

Os robôs móveis autónomos são altamente versáteis e podem executar tarefas de forma independente em ambientes desafiantes, potenciando as capacidades humanas em cenários como inspeções, vigilância e missões de busca e salvamento. Podem transmitir informações cruciais, como dados visuais, para centros de comando remoto e atuar como redes móveis de sensores para uma monitorização industrial abrangente.

Para partilhar informações cruciais e alcançar objetivos comuns de forma eficaz, é essencial estabelecer um canal de comunicação estável e fiável entre os robôs. Os canais partilhados locais de Rádio-Frequência, como o WiFi, são frequentemente mais práticos e eficientes em termos da largura de banda disponibilizada. No entanto, as soluções centralizadas de gestão de comunicações, como WiFi infraestruturado, i.e., usando Access-Points, limitam a topologia da rede. Assim, para ambientes dinâmicos, redes ad-hoc descentralizadas são preferíveis por suportarem uma topologia flexível e permitirem maior cobertura. Por outro lado, também introduzem atrasos que resultam do reencaminhamento multi-hop sempre que os robôs se afastam. Sem a devida sincronização, as transmissões de dados de alta frequência de cada robô podem causar colisões e perdas de pacotes. Portanto, é importante implementar um protocolo de sincronização e um algoritmo de roteamento responsivo que se adapte às mudanças de topologia da rede. Esta dissertação visa definir e implementar um algoritmo de roteamento para uma Rede Ad-hoc Móvel (MANET) sobre a estrutura de sincronização RA-TDMAs+, suportando comunicação *peer-to-peer*, e validando a solução encontrada num cenário prático relevante.

O algoritmo de roteamento desenvolvido, do tipo *link-state*, utiliza a informação da matriz conectividade fornecida pelo sistema TDMA para estabelecer rotas na rede. O algoritmo funciona de forma independente do controlo de transmissões TDMA usando componentes de software já existentes, realizando o encaminhamento na camada de ligação de dados (camada 2 do modelo OSI), onde apenas o endereço MAC (*Access Control Mechanism*) é modificado, enquanto os endereços IP e a informação das camadas de aplicação superiores se mantêm inalterados ao longo do caminho. A tabela de roteamento mapeia os nós acessíveis e os seus vizinhos utilizando a tabela ARP com as associações corretas entre MAC e IP. O encaminhamento é assim feito de forma automática ao nível mais baixo da pilha IP, sem intervenção das aplicações nos nodos ou de camadas de rede superiores, simplificando a gestão da rede e reduzindo o *overhead*.

Após a implementação, analisámos e avaliámos métricas de desempenho da rede e realizámos *streaming* de vídeo numa rede multi-hop com topologias variáveis. Os nossos resultados mostraram que, à medida que o número de retransmissores (hops) aumenta, também aumentam os atrasos e as suas variações. No entanto, foi possível manter uma ligação estável e realizar *streaming* de vídeo com sucesso. Além disso, ao utilizar o sistema TDMA, avaliámos o impacto das transmissões periódicas no desempenho geral da rede, fornecendo informações sobre como essas transmissões organizadas afetam a eficiência da rede.

Palavras-chave:comunicação multi-agente; MANET; redes ad hoc; robôs móveis; roteamento; sincronização; TDMA

Abstract

Mobile autonomous robots are highly versatile and can independently perform tasks in challenging environments, enhancing human capabilities in scenarios like inspections, surveillance, and search and rescue missions. They can transmit crucial information, such as visual data, to remote command centers and act as mobile sensor networks for comprehensive industrial monitoring.

To share crucial information and achieve common goals effectively, establishing a stable and reliable communication channel that uses bandwidth efficiently is essential. Local shared Radio-Frequency channels like WiFi are often more practical and bandwidth-efficient for these operations. However, centralized communication management solutions, such as infrastructured WiFi with Access Points, impose a rigid network topology. For dynamic environments, decentralized ad-hoc networks, such as ad-hoc WiFi, are preferable, offering flexible topology and enhanced coverage. On the other hand, ad-hoc networks also introduce retransmission delays caused by the multi-hop propagation between robots that lack a direct link. Without proper synchronization, the high-frequency data transmissions of multiple robots can cause collisions and packet drops. Therefore, implementing a synchronization protocol and a responsive routing algorithm that adapts to changing network topology is critical. This dissertation aims at defining and implementing a routing algorithm in a Mobile Ad-hoc Network (MANET) on top of the RA-TDMAs+ synchronization framework, enabling *peer-to-peer* communication, and validating this solution in a relevant practical scenario.

The developed routing algorithm, of the *link-sate* type, uses the TDMA framework connectivity matrix information to establish network routes but operates independently of the TDMA transmission control. It uses existing software components and performs routing at the data link layer (Layer 2 of the OSI model), where only the MAC (Access Control Mechanism) address is modified, leaving the IP address and higher-level application information unaffected. The routing table maps accessible nodes and their neighbors on the ARP table with the correct MAC-IP associations. The forwarding is then done automatically at the lower layer of the IP stack, without intervention of application software, simplifying network management and reducing overhead.

We analyzed and evaluated network performance metrics and conducted video streaming across a multi-hop network with varying topologies. Our results showed that as the number of relays increases, so do delays and their variations. However, a stable connection and successful video streaming were still achievable. Additionally, by using the TDMA framework, we assessed the impact of periodic transmissions on overall network performance, providing insights into how scheduled transmissions affect network efficiency.

Keywords: ad hoc networks; MANET; mobile robots; multi-agent communication; routing; synchronization; TDMA

Sustainable Development Goals (SDGs)

In our fast-changing world, the Sustainable Development Goals (SDGs) offer a detailed plan to create a better and more sustainable future for everyone. Established by the United Nations in 2015, these 17 goals tackle global issues such as poverty, inequality, climate change, environmental degradation, peace, and justice. As we aim to achieve these ambitious objectives by 2030, innovative technological solutions are essential in driving our progress. This chapter examines how the research aligns with the Sustainable Development Goals (SDGs). The analysis aims to identify how this work could help advance the SDGs (Table 1).

Table 1 - Alignment of the Dissertation with the SDGs

SDG	Target	Contribution	Perfomance Indicators and Metrics
9	9.1	Enhancing infrastructure flexibility and adaptability by deploying robots that operate without pre-existing infrastructure, suitable for varied and challenging environments.	Increase in efficiency and reliability of the ad hoc network.
	9.5	The MANET system leverages advanced technologies such as ad hoc networks and dynamic routing algorithms to solve complex problems and improve operational efficiency, even in areas without support.	Research and development spending on MANET technologies.
11	11.2	Autonomous robots can perform tasks such as waste management, traffic monitoring, and infrastructure inspection, enhancing urban sustainability and efficiency.	Number of urban services provided by autonomous robots (e.g., waste management, traffic monitoring).
12	12.2	Optimization of routes and synchronized communications reduce energy consumption and increase operational efficiency, promoting sustainable practices.	Decrease in waste generation through optimized operations.
13	13.1	Robots equipped with sensors monitor environmental conditions, track changes, and provide critical data to support climate action initiatives.	Number of environmental monitoring missions completed by robots.
	13.3	Assisting in disaster response and mitigation efforts.	Hability to communicate and stay in group in adverse areas.

Acknowledgements

Completing this thesis would not have been possible without the support and encouragement of several people to whom I owe my deepest gratitude.

First and foremost, I extend my sincere thanks to my supervisor, Luis Almeida, whose invaluable guidance, patience, and support were crucial throughout my research journey. His deep expertise and thoughtful advice have been fundamental in shaping this work, and I am immensely grateful for the trust and opportunity to participate in this project.

I would like to thank my co-advisor, Luis Pinto, for his valuable support and insights. His additional contributions and dedication were greatly appreciated, significantly enriching this research.

I would like to express my gratitude to all my friends and colleagues who have been with me through the challenging and joyful moments of this journey. Special thanks to Paulo for his unwavering support and motivation and to Ana Rita, who has been a constant companion since the beginning. Both of you have made this academic journey lighter and more enjoyable.

Last but not least, I express my deepest gratitude to my family. I am thankful for the unwavering support and encouragement from my mother and father, Isabel Morais and Horácio Morais, my sister, Ana Morais, and my grandmother, Maria Paulo. Thank you for your unconditional love, support, and belief in my potential. You have been the pillar of strength that sustained me through all the challenges.

Ana Morais

"You can't cross the sea merely by standing and staring at the water."

Rabindranath Tagore

Contents

List of Figures

List of Tables

List of Code Blocks

Abbreviations and Symbols

AODV	Adhoc On-demand Distance Vector Routing
ART	Active Route Timeout
BS	Base Station
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DNS	Domain Name System
DSDV	Destination Sequenced Distance Vector
DSR	Dynamic Source Routing
FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
IPC	Inter-Process Communication
LLC	Logical Link Control
LSA	Link State Advertisements
MAC	Access Control Mechanism
MANET	Mobile Ad-hoc NETworks
MPR	Multipoint Relays
MST	Minimum Spanning Tree
MTU	Maximum Transmission Unit
NIC	Network Interface Card
OLSR	Optimized Link State Routing
OSI	Open Systems Interconnection
PDR	Packet Delivery Ratio
RA-TDMA	Reconfigurable and Adaptive Time Division Multiple Access
RA-TDMA+	Ad-hoc Reconfigurable and Adaptive TDMA
RERR	Route Error Packet
RREP	Route Reply Packet
RREQ	Route Request Packet
RSSI	Received Signal Strength Indicator
RTT	Round-trip time
RTSP	Real-Time Streaming Protocol
SDGs	Sustainable Development Goals
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
WLAN	Wireless Local Area Networking

Chapter 1

Introduction

This chapter gives an overview of the contextual framework, introducing the motivations and the problem in Section 1.1, the objectives of this study in Section 1.2, the contributions of this workin Section 1.3 and finally outlines the structure of this document in Section 1.4.

1.1 Context

Mobile autonomous robots are gaining popularity due to their versatility and ability to perform a wide range of tasks. Their capacity for independent movement and advanced intelligent behaviors enables them to react and make decisions based on their perception of their surroundings, setting them apart from other, fixed, robots [2]. When working as a team, mobile robots can bring added value leveraging their collaborative capabilities, making them a valuable asset. They are effective in a variety of settings, such as severe environments, inspections, surveillance, and search and rescue missions, enhancing human sensory capabilities beyond usual geographical and safety limitations. For example, they can locate and transmit visual information about victims in catastrophic situations to a remote command center. Additionally, groups of robots can efficiently function as mobile sensor networks, e.g., overseeing extensive industrial processes and providing comprehensive visual monitoring [3].

To effectively share crucial information and achieve common goals, it is essential to establish a stable and reliable communication channel that uses bandwidth efficiently. The team-oriented operation of mobile autonomous robots often leverages a local Radio-Frequency shared channel for effective collaboration. Typical centralized communications management solutions, such as infrastructured WiFi (with Access-Points), impose a rigid network topology. However, in dynamic network environments decentralized ad-hoc networks, such as ad-hoc WiFi, can be preferred given their flexible topology and enhanced area coverage. Moreover, ad-hoc networks can also be a reliable and low-cost solution in areas where communication infrastructure is scarce or does not exist.

On the other hand, ad-hoc networks increase communication delay due to the need to forward information whenever communicating robots are not directly linked but there is a path of links

joining them. In fact, communication links between nodes can break or form due to their high mobility, but also due to nodes joining or leaving the network. This constant movement requires continuously evolving and updating communication routes between communicating robots [3]. On the other hand, each robot may transmit data with a high frequency, sharing its state or critical data with other robots, e.g., a critical video stream. Without proper synchronization, simultaneous transmissions can occur and cause collisions in the medium and thus packet drops. For better system performance, it is effective to implement a synchronization protocol for managing transmissions and a responsive routing algorithm that can adjust to the changing network topology. This approach ensures continued connectivity among all robots in the network without communication loss.

1.2 Objectives

Structuring communications over time is essential to prevent collisions in the shared medium. Relying on a global clock for synchronization can be challenging in ad-hoc networks with high mobility due to dynamic topology changes, communication delays, limited bandwidth, node failures, energy constraints, and clock drift. A possible solution is to synchronize communications without resorting to absolute time. This is the option behind the technique called Reconfigurable and Adaptive Time Division Multiple Access (RA-TDMA), which allocates predefined time slots cyclically, utilizing the reception times of packets for relative synchronization, i.e., enforcing order. The RA-TDMAs+ is the most recent protocol of the RA-TDMA family, combining the features of two previous variants, namely RA-TDMA+ [4] for ad-hoc mesh networks, and the RA-TDMAs [5], for video streaming over relay networks. This derivative protocol overlays the WiFi ad-hoc network medium access control (MAC) protocol. An experimental implementation of RA-TDMAs+ was carried out by Diogo Almeida [1] leveraging the TDMA framework developed by Luis Pinto [6].

The main objective of the work reported in this dissertation is to define and implement a routing algorithm for a Mobile Adhoc Network (MANET) built on top of the RA-TDMAs+ framework. The mechanism is designed to adapt to the functionality and constraints imposed by the framework. It aims to enhance functionality by enabling peer-to-peer communication between any nodes in the multi-hop mesh network independently of the topology at each instant. The effectiveness of the routing and peer-to-peer communication will be validated and evaluated in a relevant practical scenario under various conditions and network topologies to assess the routing robustness and adaptability.

1.3 Contributions

This dissertation presents several important contributions to the RA-TDMA framework and to routing for MANETs in general. The following contributions are outlined:

- Communication framework for teams of mobile robots that enables communication between any two nodes in the network, utilizing standard operating system interfaces, transparently to the current network topology;
- Study of routing algorithms for ad hoc networks, focusing on MANETs;
- Design and implementation of a routing algorithm specifically designed for dynamic ad hoc networks, integrated with the RA-TDMAs+ framework;
- Implementation of the ad hoc routing mechanism within the Linux operating system and using Wi-Fi ad hoc communication technology, and its interaction with the Linux protocol stack;
- Experimental test setup for dynamic ad hoc networks with topology control;
- Set of scripts to automate the deployment, execution, and processing of experiments and their results.

1.4 Document Structure

The remainder of this document is structured as follows:

- **Chapter 2** begins with an overview of the theoretical foundations relevant to the communication problem in highly dynamic mobile ad hoc mesh networks. It starts with the need for synchronization and introduces the RA-TDMA framework. Then it presents a literature review and an analysis of existing routing solutions for MANETs.
- **Chapter 3** presents a detailed explanation of the design and implementation of the proposed routing algorithm.
- **Chapter 4** explains the experimental setup as well as the metrics used for evaluation and then it describes the practical experiments that were conducted and discusses the results obtained, focusing on the performance of the routing algorithm and peer-to-peer communication.
- **Chapter 5** provides a summary of the document and a discussion on the future work.

Chapter 2

Literature Review

This chapter begins with a section that covers the fundamental concepts of wireless ad-hoc networks. It then introduces the theoretical foundation of the clockless synchronization mechanism known as RA-TDMAs+, followed by an overview of the RA-TDMAs+ protocol framework. Finally, the chapter thoroughly analyzes the routing problem and explores potential solutions.

2.1 Protocol Stack

The protocol stack is a fundamental abstraction in network design, facilitating the structured and modular development of network communication protocols. This abstraction simplifies the network communication process into manageable layers, each assigned distinct functions and responsibilities. Two primary models describe layered protocol stacks: the Open Systems Interconnection (OSI) reference and TCP/IP models [7].

The OSI model divides the stack into seven layers:

- **7 Application** – The Application Layer provides communication services directly to end-user software applications, such as web browsers, email clients, and file transfer applications. This layer allows users and software to access the network. Protocols such as HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Domain Name System (DNS) operate at this layer, facilitating data exchange and network services for various applications;

- **6 Presentation** – This layer translates data between the application layer and the network format, managing character encoding, data compression, and encryption. It ensures that data from one application's layer is interpretable by another system, regardless of differences in data representation. This layer is crucial for exchanging information among heterogeneous systems, handling data formatting and security to facilitate efficient and secure transmission;

- **5 Session** – The Session Layer manages and controls computer connections, establishing, maintaining, and terminating sessions between two communicating hosts. This layer is responsible for dialogue control and synchronization, ensuring that data streams are properly

managed and that sessions can be resumed if interrupted. It handles the setup, coordination, and termination of ongoing or interactive connections;

- **4 Transport** – This layer ensures end-to-end communication control and implements error-checking mechanisms. It manages flow control and guarantees complete data transfer, breaking large messages into smaller segments when necessary and reassembling them at the destination. Protocols such as Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) operate at this layer, offering varying levels of reliability and data integrity;
- **3 Network** – This layer processes data as packets, managing data routing, packet switching, and network congestion. It is also responsible for logically addressing devices and routing data packets from the source to the destination across multiple nodes and potentially different networks. Protocols such as the Internet Protocol (IP) function at this layer offer the necessary addressing and routing functions to ensure packets are delivered accurately and efficiently;
- **2 Data Link** – Here, the data is handled as frames. The layer provides node-to-node data transfer and handles the Physical Layer's error detection and correction. It is divided into two sublayers: Logical Link Control (LLC) and Media Access Control (MAC). The LLC sublayer handles frame synchronization, flow control, and error checking. The MAC sublayer manages the actual control of data placement on the physical medium;
- **1 Physical** – The physical layer handles data as raw bits. The data is transmitted from one physical entity to another. Physical layer protocols differ based on the type of physical medium and the nature of the signal transmitted. These signals can include electrical voltages transmitted via cables or light signals through fiber optic links, among other mediums.

The TCP/IP model divides the protocol stack into only five layers: Application, Transport, Network, Data Link, and Physical. The remaining layers (Presentation, Session) are implemented in the Application layer if necessary.

2.2 Wireless Ad hoc Network

This section covers the network technology and two foundational concepts integral to this dissertation: IEEE 802.11 (WiFi), Ad hoc networks, and Mobile Ad hoc Networks (MANETs).

2.2.1 IEEE 802.11 (WiFi)

IEEE 802.11, known as WiFi, is a set of wireless local area networking (WLAN) standards. It defines the protocols and mechanisms for wireless communication operating in half-duplex mode. An access control mechanism (MAC) is necessary to manage the efficient and fair use of the shared wireless medium. Unlike wired networks, WiFi transmitters cannot detect collisions during frame transmission, making collision avoidance crucial. Carrier Sense Multiple Access with Collision

Avoidance (CSMA/CA) is a MAC protocol used to minimize the likelihood of data collisions. Before transmitting data, a device listens to the channel to ensure it is clear. If the channel is free, the device proceeds with transmission; if busy, the device waits for a random backoff period before retrying. However, under high network congestion, CSMA/CA can face challenges like delays, increased collisions, and potential communication failures. Although insufficient to ensure collision-free synchronized communications in such conditions, the RA-TDMAs+ synchronization protocol can be implemented on top of CSMA/CA and enhance performance. In this integration, CSMA/CA helps manage unpredictable interferences, such as alien traffic or new nodes joining the network.

2.2.2 Ad hoc Network

Ad-hoc networks are decentralized wireless networks that do not rely on a pre-existing infrastructure, such as routers and access points. Individual nodes can communicate directly with other nodes or through intermediate nodes. Therefore, nodes can act as a router or a host. Determining which nodes forward data is made dynamically based on network topology. This type of network is crucial in dynamic and unpredictable environments where fixed infrastructure is impractical [8].

Ad-hoc networks are scalable, allowing the network to adapt to added or removed nodes without significant changes to the underlying communication protocol. The absence of reliance on fixed infrastructure enhances their robustness, enabling them to reconfigure in case of node failures and maintain communication links and overall system functionality. This adaptability makes ad-hoc networks ideal for large teams of robots working together. They can adjust to these changes in real time, ensuring continuous communication even as robots disperse or move to new areas.

A widely used approach to achieve synchronization in networks involves establishing a single absolute time reference, referred to as a global clock. This reference can be obtained externally, such as through GNSS or NTP over the Internet, or internally, by exchanging clock values via additional messages. However, transmitting at regular intervals can be ineffective in addressing external interference from nodes that are not part of the team or that join the network asynchronously [4]. Additionally, the mobile and ad-hoc characteristics can lead to poor performance of standard protocols, primarily due to asymmetric and unstable links, as well as the variable forwarding delays associated with asymmetric routing paths between nodes [5].

Given these challenges, it is more effective to utilize a relative synchronization protocol that synchronizes nodes based on their relative timing rather than relying on a single global clock. One of the objectives of this dissertation is to use the RA-TDMAs+ relative synchronization protocol, which is explained in Section 2.3.

2.2.3 Mobile Adhoc Network (MANET)

A MANET is a self-configuring network that operates without fixed infrastructure and comprises nodes that can move dynamically [9]. Each node in a MANET performs routing independently,

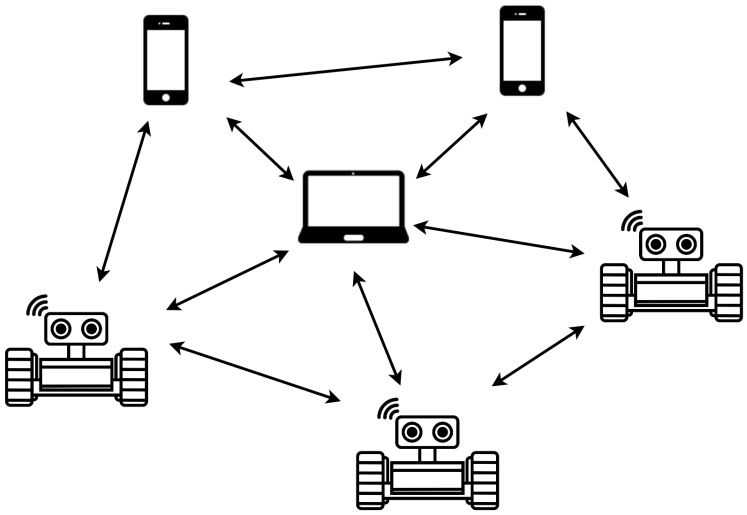


Figure 2.1 - Example of a MANET

acting as a router and a host. Due to their limited transmission range, nodes must rely on neighboring mobile nodes to forward packets. A multi-hop route connecting two nodes can be established using intermediate nodes that act as a relay, forwarding the packet to the next node in the route. This decentralized routing eliminates the potential for bottlenecks[1] [10, 9]. An example of a MANET that incorporates cellphones, portable computers, and mobile autonomous robots is represented in Fig. 2.1.

A defining characteristic of MANETs is their highly dynamic topology. Nodes in a MANET can move independently in any direction, causing the network's topology to change rapidly and unpredictably. This constant movement requires the network to have a high level of adaptability with routing protocols that can quickly respond to changes. The nodes can leave or join the system at any time. When a node joins or departs from the network, the routing tables must be updated to ensure efficient and accurate data delivery. The next hop is determined dynamically based on the current network topology [10].

MANETs also face challenges due to their dynamic and decentralized nature. Resource constraints such as limited battery life, processing power, and memory further complicate network operations, necessitating energy-efficient protocols to prolong the network's operational life. Security is another critical concern; the open nature of MANETs makes them susceptible to threats like eavesdropping and denial-of-service attacks, thus requiring robust security protocols to ensure data integrity and network reliability. Additionally, environmental factors such as interference and signal quality impact the reliability of wireless links, adding to the complexity of maintaining stable and efficient communication within the network [10].

[1] When a network is unable to handle the current volume of traffic, leading to overload, it is referred to as a network bottleneck.

MANETs have a wide range of applications due to their flexibility and ability to operate without fixed infrastructure, providing communication services in various challenging environments. Some application areas include military, remote area connectivity, disaster recovery, and vehicular communication [9].

2.3 RA-TDMAs+

In this section, we introduce the concept of TDMA and the RA-TDMA framework, and the implementations that are relevant to this dissertation.

RA-TDMAs+ is a version of the TDMA protocol that guarantees clockless relative synchronization for dynamic mesh networks. It maintains synchronization among team members, reduces interference, and overlays the CSMA/CA to create a more efficient and reliable wireless communication system. While RA-TDMAs+ defines the communication structure in the shared medium, CSMA-CA manages external interferences. This way, it is possible to modify IEEE 802.11 using standard components by controlling the transmission times from an overlay layer.

The protocol RA-TDMAs+ is a combination of two protocols. It uses the synchronization algorithm described in 2.3.3 from RA-TDMAs, and the tracking topology described in 2.3.4 from RA-TDMA+. This approach allows the implementation of the best features from each protocol, designing a robust algorithm for high-bandwidth ad hoc mesh networks.

2.3.1 TDMA

TDMA (Time Division Multiple Access) is a communication protocol that allows multiple nodes to share the same frequency channel by dividing it into equal time intervals, called slots. Each node is given a unique time slot to transmit data. The sequence of slots is called a round, which is repeated by a given period, i.e., the round period. The Fig. 2.2 represents the round period with N slots.

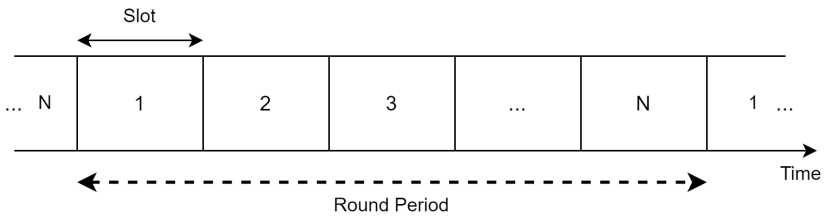


Figure 2.2 - TDMA round

2.3.2 RA-TDMA

For TDMA-type MAC protocols to work effectively, all nodes must have a synchronized perception of time. Using global clock synchronization on wireless ad-hoc is not the most efficient due to its mobile and ad-hoc nature. Therefore, it is preferable to implement relative synchronization, in which nodes use each other's transmissions to synchronize rather than relying on a clock. RA-TDMA arises from this necessity. The protocol organizes transmissions into a round, assigning one slot to each device. Each device initiates its transmission at the start of its respective slot. A packet containing the device's state is transmitted at the beginning of the slot, and synchronization is achieved based on the reception delays observed by other nodes. This protocol can overlay other communication technologies [11].

There are different versions of this protocol, but those relevant for this work are the RA-TDMA+ [4] developed for ad-hoc mesh networks and the RA-TDMAs [5] for video streaming over relay networks.

2.3.3 RA-TDMAs

RA-TDMAs [5] was developed to support video streaming between Unmanned Aerial Vehicles (UAVs) and a Base Station (BS). This protocol focuses on synchronizing nodes in a multi-hop aerial network with a distributed and adaptative mechanism without a global clock. It considers a line topology, where a node at the line's tip streams with a significant traffic load through the n hops until it reaches the base station. The authors consider the n to be limited, up to five. This limitation is due to the fact that each hop inevitably adds a forwarding delay, which influences the end-to-end latency. Each node has its own unique and sequential ID and a respective slot. At each period of time T, the node has permission to transmit packets during its designated slot. Therefore, nodes transmit sequentially, with only one node transmitting at a time. The Figure 2.3 represents an example of the referenced topology.

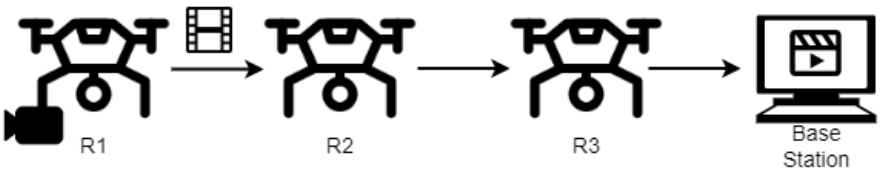


Figure 2.3 - Example of a UAVs line topology. Video streaming flows from R1 to the Base Station, using a multi-hop aerial network

2.3.3.1 Synchronization Method

The synchronization mechanism allows nodes to move their own slot boundaries to prevent overlapping transmissions. The authors state that a receiver node analyses the delay of each incoming packet. If it is identified that the packet is arriving outside of the transmitter's slot (later than expected), the receiver shifts its own slot. The receiver will now start transmitting later, thus avoiding

overlapping transmissions. If the receiver perceives packets arriving earlier than expected, it retains its slot and waits for the transmitter to advance its slot. In summary, all nodes adjust their slots to cancel out detected delays and converge to a situation where slot overlap is minimal.

Therefore, the first step is to calculate the delay of every received packet k (δ_{k}). The receiver node is identified by j and the transmitted node by i. There are n modes, each one with an equal slot length of s. For the computations, it is necessary to send in the header of the packets the slot ID and the transmission offset (p_{k}) in the transmitter's round-time frame relative to the beginning of its slot. The delay δ_{k} is calculated by the difference in the round-time of the receiver j between the real receiving time (rx_{k}) and the expected time of arrival of the respective packet k, ($\hat{rx}_k$):

$$\delta_k = (rx_k - \hat{rx}_k + T/2)_T - T/2. \tag{2.1}$$

The authors assume that the node with ID j is synchronized with its neighbor, node i. The node j has boundaries of $B^{(j)}$ and $E^{(j)}$. To determine the expected time of arrival, $\hat{rx}_k$, is necessary to find first the beginning boundary of i ($\hat{B}_k^{(i)}$) that node j would expect:

$$\hat{B}_k^{(i)} = (B^{(j)} - (j-i)s + T)_T, \quad \textit{where} \quad j,i \in [1,n] \quad \textit{and} \quad s = T/n. \tag{2.2}$$

If there were no propagation delay time, and the nodes were synchronized, the receiver j would expect the time of arrival of the packet k from node i to be:

$$\hat{rx}_k = (\hat{B}^{(i)} + p_k)_T. \tag{2.3}$$

With combining the equation (2.1) and (2.3) is finally possible to determine $\hat{rx}_k$:

$$\delta_k = (rx_k - \hat{rx}_k + T/2)_T - T/2 = (rx_k - (\hat{B}^{(i)} + p_k) + T/2)_T - T/2. \tag{2.4}$$

Now, it is possible to calculate the delay of every received packet k (δ_{k}). Before a node starts transmitting, at the beginning of its slot, all packet delay estimates saved in the previous round are analyzed, and a delay function is used to determine the shift. The defined methods to analyze the delays are the following:

Minimum delay Uses the packet with the lowest estimated delay; The slot will shift the least, making small adjustments. Though has the highest chance of overlapping with the previous slot;

Maximum delay Uses the packet with the highest estimated delay; The slot is advanced the most, thus reducing the chances for overlap with the previous slot;

Median delay Calculates the median delay for all packets, mitigating the impact of outliers. This is a trade-off between the pros and cons of the two previous approaches.

The delay is reset for the next round, as the delay function is recalculated with new values in each round. A node j can maintain its boundaries or advance (shift) the limits of its own slot according

to Δ. The final shift is determined by:

$$\Delta(r) = min(max(f(\delta_1, \delta_2, ...), 0)\Delta_max), \tag{2.5}$$

Where Δ_max represents the maximum deviation value that can be configured. The phase adjustments are bounded between zero and Δ_max (always positive). Therefore, slots can only be delayed. Additionally, the initiation of transmission for node j is set to:

$$B^{(j)}(r) = B^{(j)}(r-1) + \Delta(r). \tag{2.6}$$

Due to the linear topology, synchronization is based solely on the node in the immediately preceding slot. All nodes continuously adjust their timing until no delay is detected, ultimately achieving convergence.

2.3.4 RA-TDMA+

RA-TDMA+ [4] is designed to achieve relative synchronization in ad hoc mesh networks of mobile robots by periodically sharing their state, supporting real-time collaboration. The network is a dynamic mesh topology because only neighboring robots can communicate with each other, but it is necessary for them to communicate with the entire team. Some of the key objectives behind this protocol are to achieve synchronization without relying on a global clock and to address the lack of a direct view of team membership and network topology presented in RA-TDMA [11].This protocol is designed to overlay the CSMA/CA protocol, effectively managing external interferences such as nodes outside the team and the integration of new robots into the network.

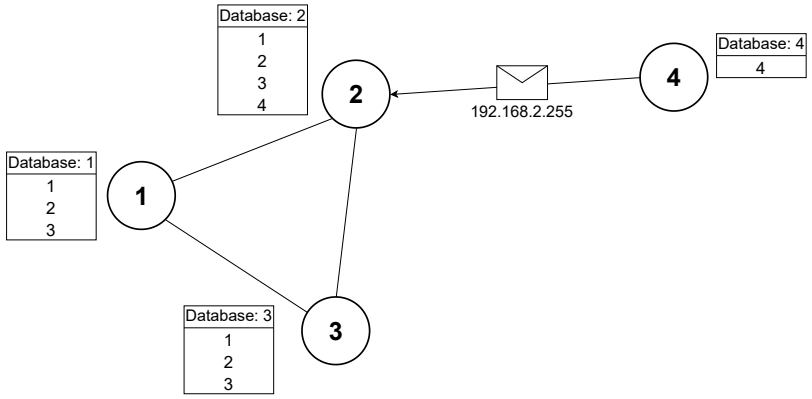
The communication is defined by a TDMA round (Fig. 2.2). The round's duration, T_{up}, corresponds to the periodicity of the communications, depending on the real-time requirements. The team set is defined by τ_{A}, and the number of active robots, N, can change dynamically as robots can enter or leave the team. Each robot has its own assigned time slot. These slots are consecutive and have the same duration ($t_{\text{slot}} = T_{\text{up}}/N$). This way, the bandwidth can be adjusted; as the number of active robots increases, the slot duration decreases.

The slots are assigned in ascending order based on the unique numeric physical ID that each robot possesses. The robot having the smallest ID is assigned the first slot of the round, which is marked by s_0. Due to the dynamic nature of the network, it is preferable to have a method to refer to the robots independently of their IDs. This approach ensures that slot assignments between robots converge to a consistent distribution as long as a stable view of the team can be maintained. As the article states, in the case where the team τ_{A} = { ID_2, ID_3, ID_5 } robot ID_2 gets slot s_0, robot ID_3 gets slot s_1, and robot ID_5 gets slot s_2 [4].

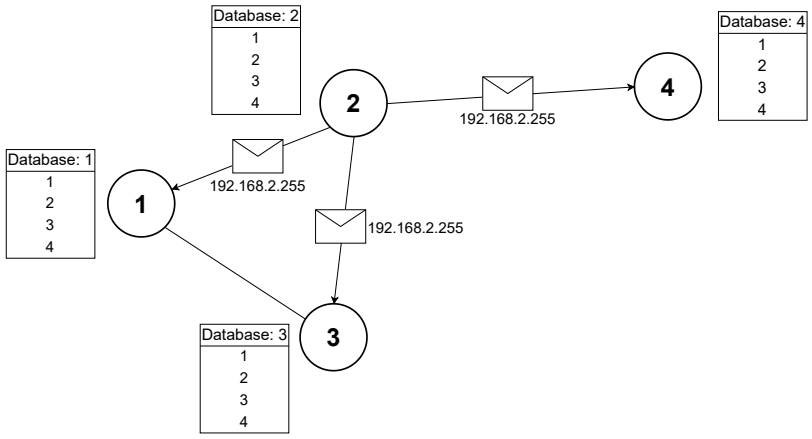
In a dynamic mesh network, there may be nodes without direct links to each other. Each robot must share its state through the mesh network to maintain a comprehensive view of team membership and network topology. This state includes the robot's current view of the topology, team membership, and other relevant information. By using flooding with aggregation, each

robot shares its state at the beginning of the slot via a multicast message. After receiving state information from neighboring robots, each robot aggregates and updates its own state. This process ensures that the team can converge to a consistent, shared state within a bounded number of rounds. In Fig. 2.4, we see the process when a new node (Node 4) joins the network. First, Node 4 broadcasts its state to its neighboring node (Node 2). Node 2 aggregates this information and updates its own state, reflecting the addition of Node 4. Next, Node 2 propagates this updated state to the rest of the network. Each node in the network, including Node 4, will eventually learn the full state of the network, ensuring all robots have consistent and updated information.

Each state variable uses an age variable to control the freshness of the information. After sharing the state, the remaining time in the slot is allocated for transmitting application messages.



(a) Node 4 joins the network



(b) The information propagates through the network

Figure 2.4 - Example of the flooding with aggregation. The nodes update their state with the new node. Node 4 can learn the network's information.

2.3.4.1 Tracking Topology

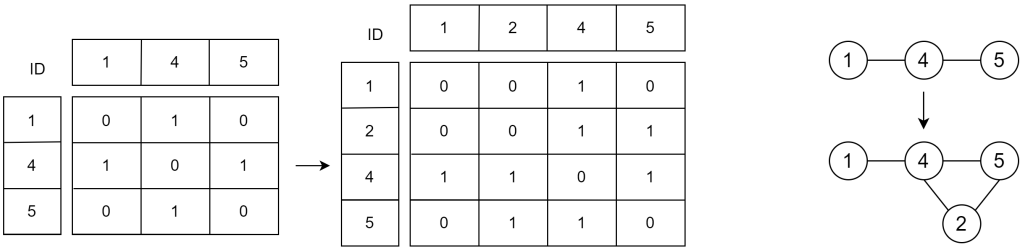
The network topology is one of the variables shared among team members. Each robot has its view of the topology, represented by a connectivity matrix M. The matrix uses a binary assignment to

represent connections, where $M(j,k) = 1$ indicates that robot l_j is receiving packets from robot l_k. Each link's state is switched after a specified number of consecutive rounds, demonstrating consistent behavior. Robot l_i must update its corresponding row i with current connections. When receiving matrices from other team members, robot l_i updates its matrix with the newest information. The matrix is continuously updated; the changes in network topology are disseminated throughout the entire network in the subsequent round [12].

2.3.4.2 Tracking Membership

Tracking membership, the team composition τ_A, is crucial for the operation of this protocol. Having knowledge of the members of the teams and their identification allows the determination of the number of active nodes N, which is essential for dividing the round into the corresponding slots and allocating the slots to the corresponding IDs. Therefore, a row in the connectivity matrix that includes each node's ID is necessary.

When a node receives a state from a joining neighbor (asynchronous), the matrix is updated with a new row and column, and N increases, creating a new slot. To ensure consistency among nodes, the round is reorganized in ascending order of ID. The new node will also receive messages from its new neighbors, updating its own matrix. The Figure 2.5 represents the update of the matrix when a new node (ID 2) joins the network, and Figure 2.6 shows the changes in the round and slot assignment upon the entering of the joining robot. The removal of a team member occurs upon the detection of one or more consecutive rounds with null rows and columns in the matrix. The updated N and the matrix are distributed throughout the network in the following round [4].



(a) Connectivity matrix **(b)** Network topology

Figure 2.5 - Update of the matrix and network topology after the entry of node ID 2 into the network

2.3.5 RA-TDMAs+: Final Overview

In conclusion, RA-TDMAs+ employs the synchronization method of RA-TDMAs, utilizing the delays of received packets to adjust the node's TDMA slot. This approach allows the network to converge to a structure with minimal overlap between transmissions, explained in the Section 2.3.3. Then, using the tracking topology and membership of RA-TDMA+, analyzed in Section 2.3.4, it is possible to apply the synchronization mechanism to a dynamic ad hoc mesh network. Every node can maintain a clear view of the network and the structure of the TDMA round.

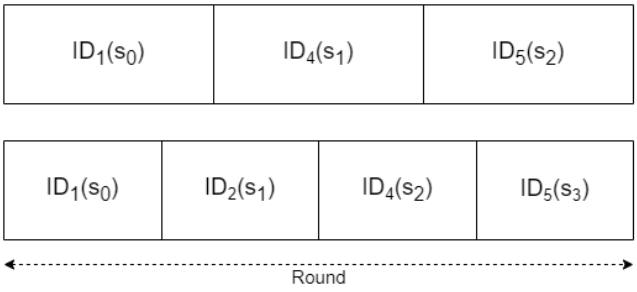


Figure 2.6 - Update of the round structure and slot allocation upon the entry of node ID 2

This way, it is possible to filter delays to avoid loops in the synchronization and then execute shifting in a highly dynamic network.

2.4 TDMA Implementation Framework

We utilized the TDMA framework for this project, initially created by Luis Pinto [6]. Diogo Almeida [1] updated the framework to a new modular, event-based TDMA architecture in user space. This architecture consists of two parts: the TDMA_Core, which handles the logic and operation of a TDMA round, and various additional modules that can be optionally used. A modular architecture allows the addition or removal of any module without disrupting the TDMA_Core operation. The modules we will be using are Topology Tracking and Synchronizer, which implements the RA-TDMAs+ synchronization protocol. The architecture is represented in Figure 2.7.

2.4.1 TDMA_Core

The TDMA module controls access to the medium. It is divided into four threads. The thread for sending packets (TX) is active during the designated slot time and will be in sleep mode for the rest of the time. The packets are organized in a queue by TDMA_Send() (TX Queue), waiting until the transmission window. As defined by the TDMA protocol, synchronization occurs just before the beginning of each slot, updating the slot boundaries if necessary. TDMA_Send() processes the packets sequentially, encapsulating each with a TDMA header before sending them via a UDP socket. The packets will be sent to their destination and received by the UDP socket of the respective TDMA module [1].

The RX thread makes the reception. The packets are decapsulated, and the TDMA module calculates the delay of the packet for future synchronization. The packets are placed in an RX Queue to be delivered to the TDMA_Receive() function for processing [1].

- **Encapsulation -** Each packet is encapsulated with a TDMA header containing critical protocol information. This header includes the packet type, the general sequence number across

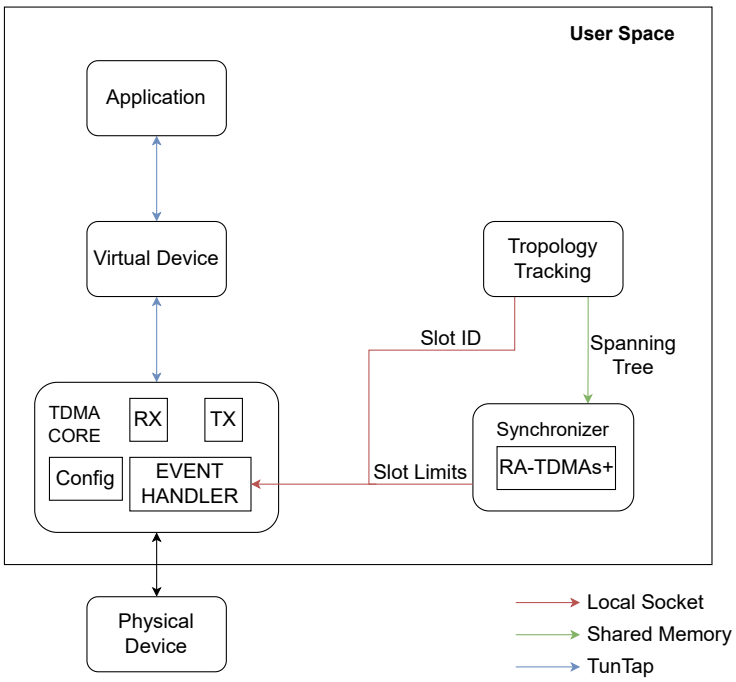


Figure 2.7 - Architecture of the TDMA framework [1]

links, the relative sending timestamp (based on the current TDMA round time), the ID, and the transmission slot limits. As noted in [1], the header is 18 bytes in size and plays an essential role in the RA-TDMAs+ protocol. The TDMA header is added on top of the application packet.

As the name implies, the third thread, Event Handler, handles the events sent to the TDMA module. Finally, the Config thread handles all the TDMA round configurations and controls the TX thread.

Each module operates as a separate process. The framework uses two types of Inter-Process Communication (IPC): AF_UNIX local sockets to enable communication between the modules and TDMA_Core, and shared memory, which allows essential information to be exchanged between modules to ensure proper operation. The TDMA_Core module creates the shared memory for the TDMA protocol, retaining exclusive write permissions. If other modules need to modify a variable, they must send a request event to TDMA_Core via AF_UNIX Sockets [1].

The framework utilizes TUN/TAP to create a virtual network interface for tunneling and network emulation, providing full control over packets sent and received on the virtual interface. The system operates in TUN mode, simulating a network layer (Layer 3) device, handling IP packets as if passing through a physical network interface card (NIC). TUN is specifically designed for IP-level traffic, unlike TAP, which works at the link layer (Layer 2) and processes Ethernet frames. This setup is ideal for network emulation and tunneling without hardware constraints.

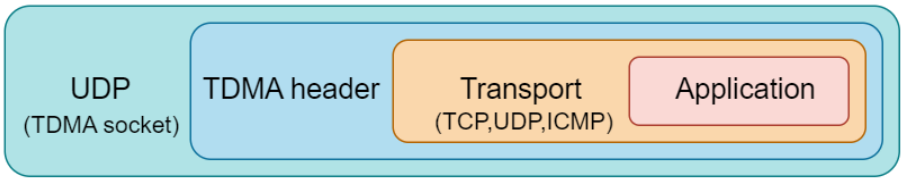


Figure 2.8 - Headers the packet has when leaving the TDMA framework. The application packet is overlaid with the TDMA header and then sent via a UDP socket.

By using the TUN interface, the system can route application-generated IP packets through user-level code. Applications send their packets to the virtual tun0 interface, which is treated as any other network device. The user-level code (*tuntap.c*) reads these packets from tun0 and passes them to the TDMA_Core.

At this stage, the user-level TDMA_Core code encapsulates the IP packets with the custom TDMA. It ensures that the packets are transmitted in their assigned time slots for efficient network communication. The TDMA_Core manages time-slot-based transmissions, encapsulating packets and forwarding them via UDP socket over the wireless ad-hoc network interface (*wlan*0) to their designated destination.

When a packet is received, the process is reversed. The TDMA_Core strips off the TDMA header, and the user-level code processes the packet. The header is essential for the operation of the RA-TDMAs+ protocol and enables the calculation of packet delays. The packet is then returned to the tun0 interface, which is treated as a standard IP packet and delivered to the application. The Figure 2.8 shows what headers a packet has when leaving the TDMA framework. The UDP packet has the Transport packet sent by the Application plus the TDMA headers as its payload [1].

2.4.1.1 Tracking Topology

This module is essential for properly operating RA-TDMAs+, and it is implemented in the file *matrix.c*. The Code Block 2.1 represents how the matrix information is saved. To maintain a direct view of the topology, it is essential to share the total number of active nodes, their respective IDs, and the connectivity matrix that contains information about the links between the nodes. Each row of the matrix has a respective *age* value that allows for the most recent information to be kept.

```
1 typedef struct{
2     uint8_t myId;
3     uint8_t numberOfActiveNodes;
4     uint8_t matrix[MNUM_DRONES][MNUM_DRONES];
5     uint8_t idOfActiveNodes[MNUM_DRONES];
6     double creationTime[MNUM_DRONES];
7     double age[MNUM_DRONES];
8 } __attribute__ ((__packed__)) tdma_matrix_t;
```

Code Block 2.1 - Variables used to save the matrix information [1]

MATRIX_init() loads the module and initializes all necessary variables. This module handles the synchronization packets sent by the other nodes in the network, which contain the matrix with topology information. MATRIX_parsePkt() parses the received packets and updates the matrix, and MATRIX_share() broadcasts the matrix to the network.

The matrix is shared when each node shares its state at the beginning of the respective time slot. The MATRIX_parsePkt() function deserializes the received packet into the variable tdma_matrix_t and uses the new information to update the local matrix. If necessary, the total number of nodes and their respective IDs are also updated, along with the order of the slots. At the end, the node updates its own line with the active links. When the ID of the respective slot changes, the module communicates the change to TDMA_Core via an event using AF_UNIX Sockets.

At the beginning of the time slot, before MATRIX_share() transmits the local information to the other nodes in the network, dead links and their associated node IDs are removed from the matrix. If a row for a certain node ID has an age greater than 3 rounds, the node is considered to have left the network and no longer has a connection. It is then necessary to update the slot IDs and inform the TDMA_Core. Afterward, the age of each line is updated, and the matrix is serialized, as shown in Figure 2.9, and broadcast through the network. The matrix represents the network topology in binary form. A value of 1 indicates a connection between nodes, while 0 indicates no connection.

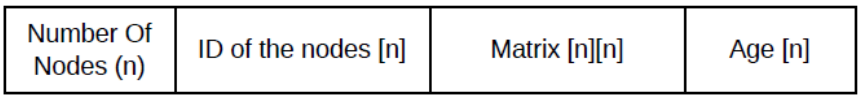


Figure 2.9 - Structure of the Matrix Packet sent [1]

A Spanning Tree is generated from the connectivity matrix using the Prim algorithm and then shared with other modules. When the matrix needs to be sent, the Spanning Tree is updated after removing links timed out and updating the local matrix. The variables stored in the Spanning Tree's Shared Memory are shown in Code Block 2.2.

```
1 typedef struct{
2 pthread_mutex_t mut_spanningTree;
3 uint8_t numberOfActiveNodes;
4 uint8_t spanningTree[MNUM_DRONES][MNUM_DRONES];
5 uint8_t idOfActiveNodes[MNUM_DRONES];
6 } spanningTreeSHM_t;
```

Code Block 2.2 - Variables saved in the Shares memory relative to the Spanning tree [1]

2.4.2 Synchronizer

This module implements the RA-TDMAs+ synchronization algorithm. It has access to the delays calculated by TDMA_Core and the Spanning tree to keep track of the topology, both saved in Shared Memory. The module is in sleep mode until three milliseconds before the end of the TDMA round. In this window, the RA-TDMAs+ accesses the delay values, determines the δ_k value, and returns this information to the TDMA_Core. This information must be calculated and sent before TDMA_Core changes the slot limits at the end of the round. The slot is adjusted based on the determined delay delta value, limited to twenty-five percent of the respective slot. The RA-TDMAs+ adjusts the slot limits with the new value and sends it to the TDMA_Core with an event [1].

The Spanning Tree allows filtration of the delays, avoiding topology loops in the synchronization algorithm. Only delays corresponding to links present in the Spanning Tree between the transmitting node and the synchronization node are considered. Once filtered, the delays are categorized by transmitter ID, and the average delay for each ID is calculated. This procedure is the procedure detailed in 2.3.3.

The Δ_max parameter, set during the configuration phase, defines the maximum allowable phase adjustment for a slot in each round, preventing significant shifts in its position. This parameter plays a crucial role in managing the slot adjustment process: it accelerates convergence to a global virtual TDMA round when the adjustment exceeds the set limit and slows it down when the adjustment is below that limit [5].

Therefore, for slot adjustment, the Δ_max value is determined, constrained to a maximum of twenty-five percent of the current slot size. The slot boundaries are then updated with this new delta value and sent to the TDMA_Core using an event [1]. If the calculated delay surpasses this threshold, the maximum delay experienced by the slot is capped at this specified limit. This module operation is presented in Figure 2.10.

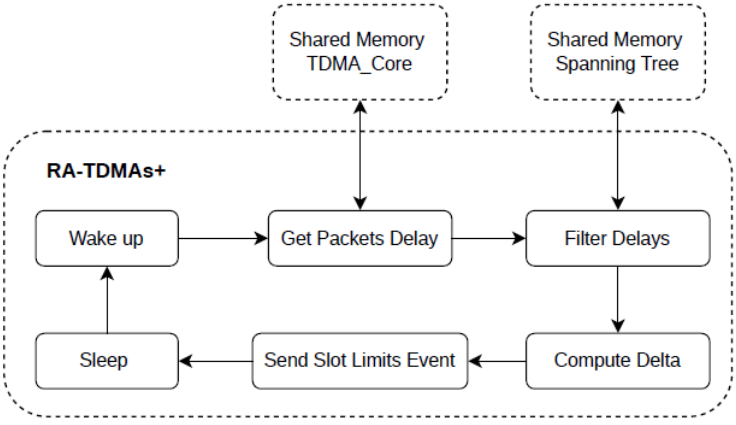


Figure 2.10 - Stages of the Synchronization module operation [1]

2.5 Routing Algorithms

The research study will use a team of small mobile robots as the testbed. Each robot will be a node in a wireless ad hoc network, which will be arranged in a mesh topology. The robots can move freely within the network, creating dynamic connections as needed. These features align with the MANET description in Section 2.2.3, as the small autonomous mobile robots will establish a self-organizing network.

In a highly dynamic ad hoc network, it is necessary to develop an algorithm that can adapt quickly to topology changes and maintain efficient routing paths to enable reliable and robust communication between any two nodes.

From the classical categories presented, protocols evolve in different branches, such as Geographical, Multicasting, Hierarchical, and others. Our focus will be on the topology-based routing protocol options. These protocols use network topology information to make routing decisions. They can maintain and update the network's structure, including nodes and their connections, to ensure efficient data packet delivery from source to destination. It is possible to classify them into three categories, as follows.

Reactive Routing Referred to as on-demand, these protocols create routes only when a packet needs to be sent. There are no fixed routes. If a route to a specific destination is unavailable in the cache, the protocol initiates a route discovery process to establish a new path. The best route is found through requests for route information sent in broadcast [13]. When the path is found and updated, the packet is forwarded. Reactive protocols generate minimal network overhead but incur delay time during route discovery [14].

Proactive Routing Also known as table-driven protocols, it relies on maintaining routing tables of known destinations and details about the network's topology. With these protocols, it is possible to forward the packets immediately. However, this implies that the routing tables must be updated periodically. Therefore, it is necessary to trade frequent update messages between neighbors to keep the information about the network fresh. [14].

Hybrid Routing Hybrid protocols combine reactive and proactive routing protocols. The objective is to reduce the control traffic overhead caused by frequent updates between nodes from proactive systems while reducing the route discovery delays of reactive systems by preserving a routing table in some form [13].

Protocols use three common techniques to route packets: source routing, distance vector routing, and link state. In source routing, routing information is included in the packet headers, which means nodes do not need to maintain routing databases. However, this method results in high network overhead. On the other hand, distance vector routing uses next hop and destination addresses to route packets. Nodes need to store active route information until it is no longer necessary or until an active route timeout occurs to prevent outdated routes [15]. Finally, the link state is a

routing algorithm that builds and maintains a comprehensive network map. Each node in the network maintains detailed information about the state of its links and exchanges this information with other routers [16].

The objective is to implement a routing algorithm that overlays the synchronization protocol RA-TDMAs+. This section will focus on comprehensively analyzing and understanding existing routing protocols for MANETs. This analysis aims to provide a deeper understanding of routing protocols designed for dynamic networks and assess how these protocols can be integrated or inspire the development of a routing algorithm for the RA-TDMAs+ framework. Our focus will be on the reactive and proactive topology-based routing protocol options. The Figure 2.11 presents the chosen protocols.

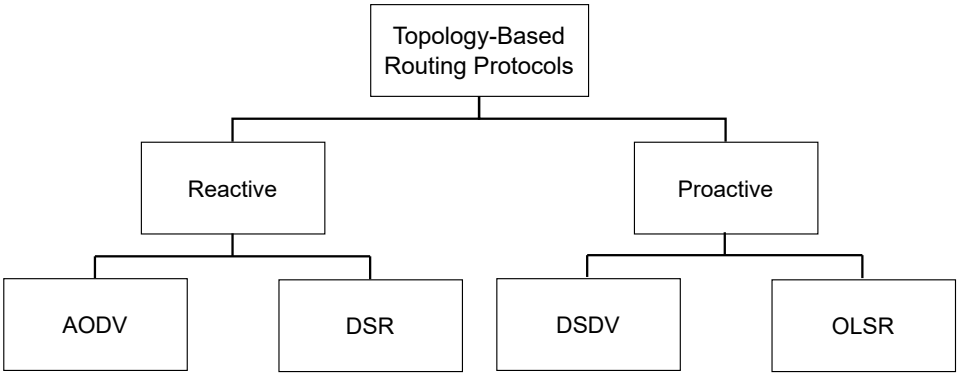


Figure 2.11 - Topology-based routing protocols

2.5.1 Destination Sequenced Distance Vector (DSDV)

DSDV is a proactive protocol; therefore, all nodes have stored a routing table with routes to every node in the network [17]. This proactive protocol was proposed by [18] and uses the distance vector algorithm. It utilizes the Bellman-Ford algorithm to determine the shortest number of hops to reach the destination.

To update the routing information, each node needs to advertise its information with its current neighbors by broadcast or multicast. Each node in a network has a unique sequence number assigned to it. This sequence number determines the age and relevance of each node in the routing information stored in the table. Whenever there is any change in the routing information, the node increases its sequence number to notify its neighbors that the information has been updated. When a node receives an update, it compares the sequence numbers of the information it has with the one it just received and decides whether to update its routing table or not [19]. The advertisements share the following information:

- Destination IP address;
- Number of hops required to reach the destination;
- Sequence number of the information received regarding that destination [19].

The use of sequence numbers helps prevent loop formation and ensures that the most recent and accurate routing information is used, avoiding stale routes. The information can be shared with a full dump, where all available routing information is sent, or an incremental carry, where the message sent only has the changes in the routing table since the last dump. This way, it is possible to reduce the overhead caused by the periodic updates [19].

If a node does not get updates from a neighbor for an extended period, it assumes the neighbor is unavailable or has experienced a link failure. In such circumstances, the node increments its own sequence number and broadcasts the alteration, declaring the associated routes invalid. However, when a link or route becomes unreachable, it may take some time for this information to propagate through the network, and during this time, nodes may incorrectly believe that the route is still valid. This problem is known as the count-to-infinity problem [18].

The periodic updates can drain battery power and consume bandwidth even when the network is idle. Also, when there is a change in the network, is necessary to update the sequence number before the network re-converges. This fact indicates that DSDV is not suitable for highly dynamic networks [19].

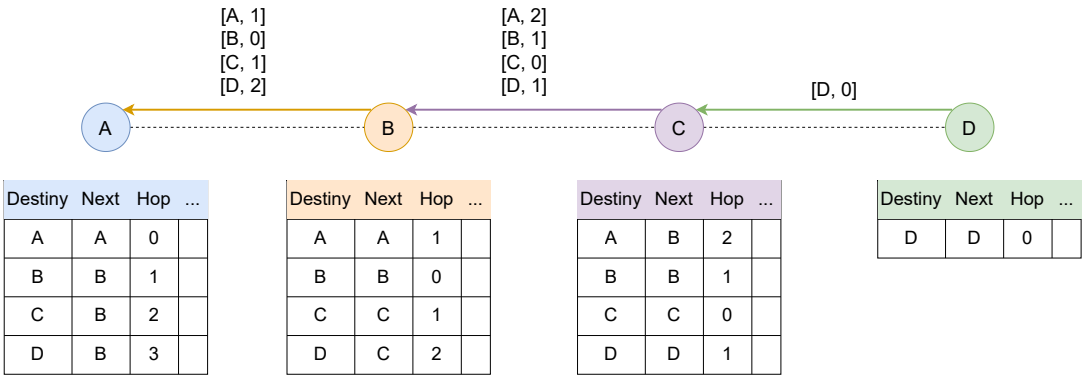


Figure 2.12 - Example of DSDV routing with four nodes (A, B, C, D). When node D joins the network, the information is propagated throughout the network.

The Figure 2.12 demonstrates how the DSDV protocol operates, with nodes exchanging distance-vector updates to maintain fresh and loop-free routes. Each node continuously updates its routing table based on information received from its neighbors. When Node D joins the network, it shares its routing information (which consists only of itself) with Node C. Node C then updates its routing table and propagates the updated information to Node B. This process continues throughout the network, allowing all nodes to learn about the addition of Node D. Similarly, Node D will receive routing information from Node C and update its own table to include paths to other nodes in the network.

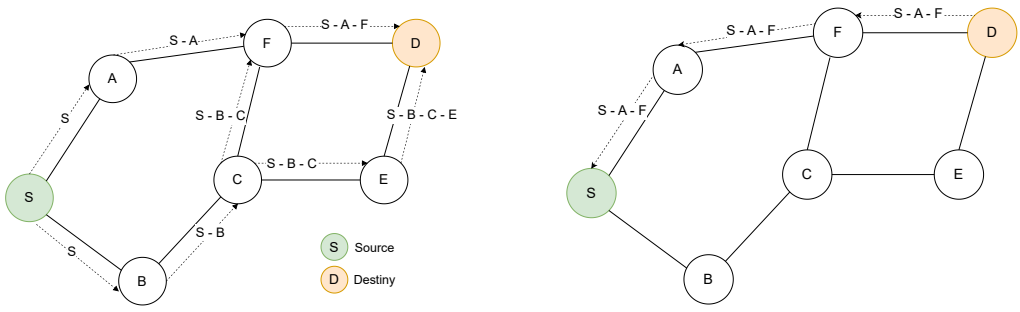
2.5.2 Dynamic Source Routing (DSR)

Dynamic Source Routing is a reactive protocol. This protocol eliminates the need for network infrastructure or administration, as these networks are entirely self-configured and self-organized.

DSR uses source routing for packet forwarding, where the source node specifies the complete route that a packet must follow. Intermediate nodes do not need to maintain routing information for the packet's destination [17].

When a source node needs to send a packet, it initiates a route discovery process by broadcasting a Route Request Packet (RREQ) throughout the network. When this request reaches the destination node or an intermediate node with a cached route to the destination, a Route Reply Packet (RREP) is sent back to the source node with the full route information. Using this technique, it is possible to have multiple routes to a destination since the source node may receive several RREP messages that flood from different routes. Then, the protocol chooses a route based on a defined metric (e.g. number of hops) and stores the route temporarily in its cache [19]. As shown in Fig. 2.13, the source node S broadcasts the RREQ to its neighbors, and the request propagates through the network, accumulating the path it traverses. When the RREQ reaches the destination node D, it responds with an RREP, sent back to the source node along the reverse path. In this example, node D selects the path "S-A-F-D" for the RREP, allowing node S to use this path for future communication.

Route maintenance involves monitoring the route for correct operation, given that a neighbor node might move or fail, rendering the route obsolete. The authors of DSR [20] explain that while a route is being used, the route maintenance procedure constantly checks the route's operation and notifies the sender of any routing errors. The Route Error Packet (RERR) contains the addresses of both hosts: the host that detected the error and the host it failed to transmit to on this hop. Then, the sender updates its routing cache and looks for a new one.



(a) Process of RREQ **(b)** Process of RREP

Figure 2.13 - Example of DSR process. The left diagram shows the RREQ phase, where node S discovers multiple potential routes to node D. The right diagram illustrates the RREP phase, where node D selects a route and replies to node S, establishing a source-routed path.

2.5.3 Adhoc On-demand Distance Vector Routing (AODV)

AODV is another reactive protocol that uses hop-by-hop routing and initiates route discoveries only when routes are needed, performing route discovery using on-demand Route Requests Packets(RREQ). Similarly to the route discovery process of DSR, in Section 2.5.2, the RREQ floods the networks until it reaches a node that knows the route to the destination, or it is the destination

itself. Then, the designated node answers with an RREP that flows for the reversed path set up by the RREQ. The critical difference between both protocols is that AODV uses distance vector routing while DSR uses source routing. This technique requires all intermediate route nodes to store the designated route. Therefore, each node in the network must save all routes in its cache that it participates in until they are no longer necessary or until a timeout occurs [19].

The AODV authors [21] explains that it uses the sequence numbers and routing beacons from DSDV. AODV provides loop-free routes by each node maintaining a monotonically increasing sequence number. These numbers are included in the RREQ and RREP packets. Nodes use these sequence numbers to determine the freshness of routes and avoid stale information. The nodes' routing tables in the neighborhood are optimized for quick response time to local movements and establishing new routes.

Routes are associated with Active Route Timeout (ART), an expiration time. If a route is not used for some time, it gets removed from the routing table as it is considered stale. If a link or node fails, AODV triggers a Route Error (RERR) message, notifying affected nodes that the route is no longer valid. Subsequently, affected nodes can initiate a new route discovery process to find an alternative path.

2.5.4 The Optimized Link State Routing Protocol (OLSR)

OLSR optimizes the traditional link-state protocol designed for mobile ad hoc networks. The exchange of topology information with other nodes in the network occurs regularly in a table-driven manner, with hop-by-hop routing. Each node selects a group of neighbor nodes as "multipoint relays" (MPR), responsible for forwarding control traffic throughout the network. The MPRs are the only ones in charge of forwarding control traffic dissemination throughout the network. MPRs provide an efficient mechanism for controlling flooding traffic by reducing the required transmissions. To select MPRs, a node chooses its one-hop neighbors with a symmetric, i.e., bi-directional, linkage. MPR nodes declare link-state information for their MPR selectors, allowing OLSR to give the shortest path routes to all destinations [16].

When the communicating pairs change, no extra control traffic is generated. Routes are always maintained for all known destinations. OLSR relies on periodic Hello messages exchanged between neighboring nodes to detect the presence and status of neighbors. These messages help maintain a consistent network topology view by identifying active and reachable nodes [16]. The authors [22] explains that after selecting the MPRs, the messages exchanged contain information about neighboring nodes and the quality of links to them. These are called Link State Advertisements (LSA). Each node constructs its routing table using the Dijkstra algorithm based on the received LSAs. This algorithm enables the node to determine the shortest paths to all other nodes in the network, enabling it to determine optimal routes. The routing table is recalculated when a change in the neighborhood is detected regarding a bi-directional link or when a route to any destination has expired.

In the network topology shown in 2.14, the source node (black) communicates with its one-hop neighbors (orange) by selecting MPRs (blue) to forward messages to two-hop neighbors (white).

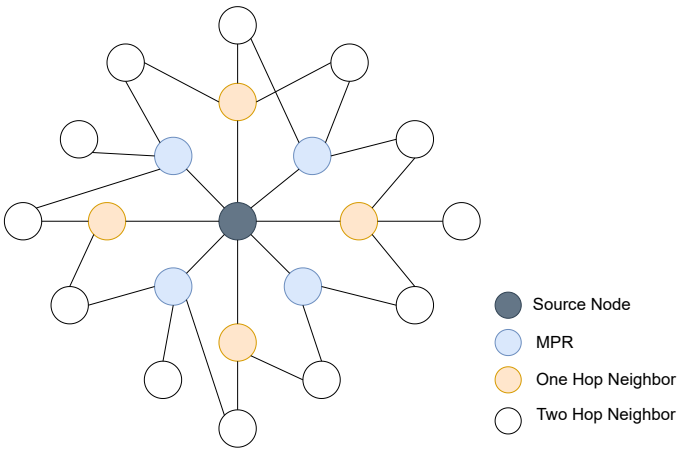


Figure 2.14 - Network topology illustrating Multipoint Relay (MPR) selection. The source node (black) communicates with one-hop neighbors (orange), while MPR nodes (blue) relay messages to two-hop neighbors (white).

Instead of flooding the network, only the MPRs relay the message, reducing redundant transmissions. This selection process optimizes communication by ensuring efficient message delivery to all nodes while minimizing network congestion, making it especially useful in larger, more complex networks.

2.5.5 Analysis of Routing Algorithms for RA-TDMAs+

The highly dynamic topology imposed by the mobile team of small robots necessitates constant tracking and updating of the routing matrix. Therefore, utilizing the state exchanges already established by RA-TDMAs+ is beneficial. The topology information shared in these state exchanges aids in implementing a proactive protocol, as periodic exchanges of network information are already in place. Consequently, additional control traffic overhead is unnecessary.

Each node shares its connectivity matrix at the beginning of its slot. This matrix is continuously updated, providing a direct network topology view. Consequently, the routing algorithm will have access to the latest network topology updates. Additionally, it will be capable of providing information about all known destinations and the network topology, being updated periodically, which characterizes it as a proactive protocol. If every node in a multi-hop network shares the same network topology view, routing can be implemented using a consistent and unified method.

The connectivity matrix will be utilized to our advantage, being the routing basis of the algorithm. This algorithm will be proactive, with no route discovery, as the connectivity provides information about the complete network. This allows quick adaptation to changes in the network, as the RA-TDMAs+ already handles the propagation of changes.

The intention of using a routing algorithm to overlay the RA-TDMAs+ introduces specific considerations that must be taken into account, which influence the implementation of an algorithm. Although RA-TDMAs+ enables collision-free communication in a clockless synchronized

manner, it is important to note that a node must wait for its designated TDMA slot to share information. This implies that any transmissions will be interrupted and delayed until the designated transmission slot returns. The delays will depend on the round's organization and the nodes' respective IDs involved in the route. When a route has an organization that differs significantly from the organization in the TDMA round, more rounds will be required to forward information.

The size of the network impacts both RA-TDMAs+ and the routing algorithm. As the network grows, the number of TDMA round splits increases. With a fixed period (T), the slot size decreases, leaving less time for transmissions. Additionally, as the number of nodes increases, the complexity of operations related to the connectivity matrix also rises, affecting both its updates and the routing algorithm. This results in increased computation time. Therefore, at the beginning of each node's respective slot, it is necessary to compute the delays (as explained in Section 2.3.3.1), update the matrix for sending, and, if necessary, update the routing matrix. If these operations are too time-consuming, the rest of the available packet transmission windows might not be sufficient, as fewer packets will be transmitted in each slot. In conclusion, routing performance depends not only on network factors such as size and complexity but also on the conditions and characteristics imposed by RA-TDMAs+. Furthermore, we require a protocol that is fast and simple enough to execute with minimal computational overhead. The goal is to update routing during the RA-TDMAs+ processing time at the beginning of the transmission slot, minimizing delay for route updates during the transmission window.

Analyzing the referred protocols is essential to learning and understanding established techniques for routing in MANET environments, which can later be applied or adapted to meet the framework's requirements. Implementing the route discovery feature of reactive protocols would not be advantageous. Apart from the fact that the complete view of the network is already shared between the nodes if a node tried to initiate a route discovery, it would have to wait for each node's designated slot to arrive before the responses could flow through the network, which could negatively impact performance.

The Table 2.1 provides an overview of the mentioned protocols, and Table 2.2 offers a brief analysis of the protocols based on the desired deployment environment.

Table 2.1 - Overview comparison between the routing protocols analyzed

Feature	**DSDV**	**DSR**	**AODV**	**OLSR**
Type	Proactive	Reactive	Reactive	Proactive
Philosophy	Distance Vector	Source Routing	Distance Vector	Link State
Discovery	Periodic updates	On-demand	On-demand	Periodic updates
Maintenance	Sequence Numbers	On-demand	On-demand	Periodic Updates

Table 2.1 - Overview comparison between the routing protocols analyzed (continued)

Feature	**DSDV**	**DSR**	**AODV**	**OLSR**
Control Overhead	High, due to periodic updates	Lower, only when needed	Lower, only when needed	High, due to periodic updates
Scalability	Moderate	Limited	Moderate	Good
Latency	Low	High (due to route discovery)	Moderate(due to route discovery)	Low
Information Storage	Every node saves all routes	Each packet carries the complete route. Nodes do not store routes	Each packet carries the complete route. All nodes save routes	Link-state database. Each node has a complete view of the network
Mobility Handling	Poor	Good	Good	Moderate

Table 2.2 - Analyzes of the routing protocols

Protocol	**Analyses**
DSDV	This protocol needs periodic transmissions to share the routing table between the nodes in the network. RA-TDMAs+ already has periodic transmissions. If we decided to implement DSDV, sharing its routing table alongside the RA-TDMAs+ information at the beginning of the slot would result in higher overhead. However, sharing the connectivity matrix might be sufficient to update the routing table. Instead of sharing the routing table directly, nodes could use the connectivity matrix to determine and update their routing tables. The *age* variable in the connectivity matrix would ensure that the most recent and accurate information is used. The Bellman-Ford algorithm could be employed to determine routes based on the matrix. Each node would maintain a routing table with the destination and the number of hops required. Knowing the number of hops can benefit other implementations, such as path selection and load balancing. This protocol has characteristics that could integrate well with the framework and assist in designing the desired routing algorithm.

Table 2.2 - Analyses of the routing protocols (continued)

Protocol	Analyses
DSR	Considering that the framework may not perform optimally with packet-based route discovery, there are still valuable characteristics to consider when applying the routing algorithm. This protocol focuses on source routing, where middle nodes don´t need to keep information about routes, as the routing information is included in the packet headers. The source node can determine the route from the connectivity matrix and include it in the packet header. Middle nodes receive the packet, check the header, and forward it to the next hop or the destination. To ensure the packet reaches its destination despite errors or route changes, route maintenance must also be implemented. In this approach, nodes do not maintain a routing table but calculate the necessary route only when sending a packet. If a stream is sent over multiple rounds (because it wasn't possible to send everything in one slot), there will be updates to the matrix, resulting in updates to the route used. With each round, packets from the same stream might follow different routes if there are topology updates. However, they can only follow routes known to the source node, so if the source node doesn't have an up-to-date and synchronized view, there will be more errors when sending packets
AODV	This protocol functions similarly to DSR but offers greater advantages due to its use of distance routing. While DSR relies solely on the source node's view and is more prone to inconsistencies and errors if the node is not synchronized, AODV requires all nodes on the route to store routing information. In this approach, the source node includes the route in the packet header and forwards it. Middle nodes receive the packet, save the routing information, and forward it to the next node. If an error or problem is detected, the node can recalculate the route using its connectivity matrix. This feature is particularly useful because if the implementation requires middle nodes to wait for their transmission slot, they can access a more recent version of the topology and make necessary corrections.

Table 2.2 - Analyses of the routing protocols (continued)

Protocol	Analyses
OLSR	Each node already shares its state at the beginning of transmissions, performing the same function as the Hello messages in OLSR. In their state, nodes share connectivity information, allowing them to update their view of the network, similar to a link-state protocol. While RA-TDMAs+ enables information sharing via broadcast, OLSR uses MPR selection to reach all 2-hop neighbors. The objective of MPRs is to optimize the flood of control messages required to disseminate topology information. The connectivity matrix makes MPR selection more accurate and efficient. However, even though MPR can reduce the number of transmissions and overhead, RA-TDMAs+ requires periodic sharing of node states. Within the same message, it handles forwarding the same topology view to all nodes and detects links. When a node receives this state message, it can confirm a direct link, update its matrix, and verify the rest of the links and active nodes that it might not have a direct view of. Furthermore, each node is restricted to transmitting only in its slot. Therefore, using MPR to forward information might not offer the best performance in this framework, especially with a small number of active nodes.

2.6 Summary

This chapter addressed both the theoretical and practical aspects that are crucial for this dissertation. We examined the Wi-Fi networking technology, focusing on the characteristics of the ad hoc mode and the testbeds designed for MANETs. Next, we explored the synchronization protocol, RA-TDMAs+ and the framework in which it is implemented. This framework will later be enhanced with necessary modifications to support the routing algorithm we will develop and to address the communication needs between any two nodes in the network. Finally, we analyzed various existing topology routing algorithms for MANETs, providing a detailde description of their operation, applicability, and functionality within the intended framework and testbed.

Chapter 3

Routing Algorithm: Design and Implementation

This chapter outlines the stages of the methodological process. Although the routing protocol is specifically designed to work effectively with the RA-TDMAs+ protocol framework, it can also operate independently. The chapter proceeds to detail the steps involved in defining and implementing the routing protocol.

The routing algorithm is specifically designed to optimize the operation of TDMA, leveraging its advantages while adapting to its constraints. While the routing module coordinates with TDMA, it operates independently of the topology management and transmission control mechanisms that TDMA provides. This independence ensures that the routing algorithm maintains consistent performance in setting routes, regardless of the underlying TDMA structure. However, when TDMA is utilized, the performance related to communications is conditional upon its operational structure and constraints.

The routing code is implemented using the modular architecture, allowing it to run in parallel with the TDMA_Core module without interrupting or interfering with its operations.

3.1 Designing the Routing Method

The algorithm takes advantage of the constant sharing and updating of network topology provided by the TDMA framework. In each round, each node shares its connectivity matrix with neighboring nodes. When the node is prepared to transmit and share its current state, it sends its most up-to-date network view. This occurs after the node receives and incorporates matrices from all its neighbors in the previous round and updates its local topology by identifying and removing timed-out links. This topology information is stored in a [N][N] connectivity matrix, where N represents the number of active nodes in the network. The matrix describes all the connections, including possible paths and even loops between nodes.

To choose which packets to use for synchronization, RA-TDMAs+ creates a Spanning Tree using Prim's algorithm. This Spanning Tree is updated whenever the connectivity matrix is refreshed, including when the matrix is prepared for transmission. Also, the updated Spanning Tree is stored in shared memory, allowing other modules in the system to access the current network topology. In this approach, using the Spanning Tree becomes advantageous, as the new routing module can access the most up-to-date topology information, ensuring an already loop-free network structure.

3.1.1 Prim's Algorithm

Prim's algorithm is a greedy method used to find the Minimum Spanning Tree (MST) of a connected, undirected, and weighted graph. The MST connects all vertices with the smallest possible total edge weight, ensuring no cycles are formed. However, the path between any two nodes in the MST may not necessarily be the shortest path between them in the original graph.

Prim's algorithm begins at an arbitrary vertex. It progressively adds the smallest edge that connects a vertex in the MST to one outside of it, expanding the tree until all vertices are included. When implemented using an N x N adjacency matrix, where N is the number of vertices, the algorithm evaluates each vertex and its neighboring edges to find the minimum. This results in an $O(N^2)$ time complexity, as the algorithm must evaluate all edges and vertices to construct the optimal MST.

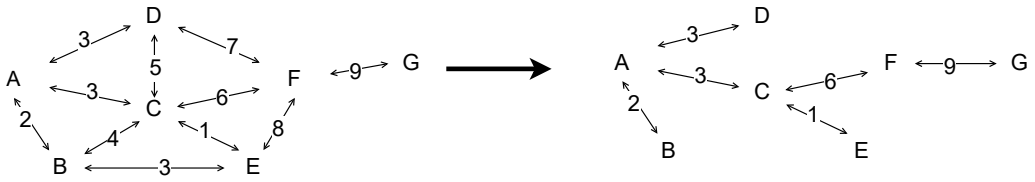


Figure 3.1 - Example of an application of the Prim's Algorithm. The values represent the weights of the links.

- **Other Algorithms**

Dijkstra's algorithm is also a greedy approach. This one focuses on finding the shortest path from a source vertex to all other vertices in a graph with non-negative edge weights. The algorithm starts by selecting the source vertex, then iteratively chooses the vertex with the smallest tentative distance, updating the shortest paths to its neighboring vertices. In the context of routing mechanisms, Dijkstra's algorithm determines the most efficient paths through a network, optimizing data transmission routes.

When implemented using an N x N adjacency matrix, where N is the number of vertices, the algorithm has a time complexity of $O(N^2)$, as it must evaluate all vertices and edges to correctly update the shortest paths from the source node.

In relation to the state-of-art analyzed in 2.5, the DSDV utilizes the Bellman-Ford algorithm to determine the routing tables by finding the shortest path from a single source vertex to all other vertices in a graph. Unlike Dijkstra's algorithm, Bellman-Ford can handle negative edge weights. It works by iterating (N-1) times, where N is the number of vertices, checking each edge to see if a shorter path can be found. This process ensures that the shortest paths are identified.

When run on an N x N binary connectivity matrix, where "1" indicates a connection, it minimizes the number of hops by treating all edges as having equal weight (1). The time complexity is $O(N^3)$, as it iterates through the entire matrix for each N-1 iterations.

- **Conclusions**

Although Dijkstra's and Bellman-Ford's algorithms are commonly used for routing by calculating the shortest path, their implementation requires processing the connectivity matrix. To achieve this, it is necessary to create a structure similar to a Spanning Tree (*spanningTreeSHM_t*) in shared memory, allowing the essential information to be exchanged with the routing module to execute the algorithm. Both Dijkstra's and Bellman-Ford's algorithms are moderate to complex regarding time and processing.

It is, therefore, more advantageous to use the existing Spanning Tree, which is already shared and executed as part of the system. It can then be used to compute the routes. This approach helps reduce the time complexity of execution and processing, ensuring that routes are calculated as quickly as possible to achieve optimal transmission performance within the slot time. Prim's algorithm prioritizes link quality to ensure a balanced and stable network. While this method may result in more hops between nodes, it emphasizes link reliability and maintains balance across the network, leading to a more dependable and resilient topology.

Currently, the connectivity matrix is represented in binary form, with no additional cost associated with link quality. As a result, the Minimum Spanning Tree (MST) outcome depends on the order of the nodes and the sequence in which the algorithm is executed. In future work, incorporating link quality values could enhance the accuracy and rigor of the topology determination process, leading to more reliable network formation.

The aim is to use the information provided by RA-TDMAs+ to avoid creating additional traffic and overhead while determining routes. The algorithm uses the information saved in the *spanningTreeSHM_t* structure, shown in the Code Block 2.2. All the necessary information is included, including the topology, number of active nodes, and respective IDs.

The routing algorithm is based on topology sharing, where each node calculates its routing table locally. By continuously maintaining an up-to-date network view, all neighboring nodes converge on the same routes, ensuring consistent and efficient data transmission.

In relation to the study conducted in Section 2.5, the routing algorithm will resemble DSDV by maintaining a routing table that includes routes for all nodes in the network. Additionally, like OLSR, it will function as a link state routing algorithm, allowing all nodes to share a comprehensive view of the network topology.

3.1.2 Routing Matrix

Once the network topology has been established using Prim's algorithm to form the Spanning Tree, the result is a loop-free structure where each node has a single path to every other node. The next step is to convert this topology into a routing matrix. This matrix must accurately represent the necessary information for the system to implement and manage the routes efficiently. The routing matrix will serve as the basis for guiding data packets through the network, ensuring that the routes are aligned with the paths determined by the Spanning Tree.

Each node will generate its local routing table, where all nodes converge to the same routes. Having information about the active nodes in the network, it is possible to run through the spanning tree and generate paths for each node.

Following the example shown in Figure 3.1, the routing matrices for the corresponding source node of A, C, and G should be as shown in Figure 3.2, respectively. It is important to note that the weights represented in Figure 3.1 are for illustrative purposes only; in the used framework, weights are not utilized. The spanning tree is generated by running the binary connectivity matrix. For source node A, it can be confirmed that it can communicate directly with nodes B, C, and D. Through these nodes, particularly C, node A can reach the remaining network members, E, F, and G. This same logic applies to each node within its local framework. For example, node G illustrates a case where, if a node can only communicate with a single neighbor, that neighbor becomes responsible for retransmitting G's packets when communicating with other nodes.

A	
B	-
C	E F G
D	-

C	
A	B D
E	-
F	G

G	
F	C A B D E

Figure 3.2 - Example of the definition of a routing matrix following the example of the spanning tree in Figure 3.1

3.2 Implementing the Routing Method

To guarantee peer-to-peer communications, packets must be transmitted with their original source and destination IP addresses intact. Preserving this information ensures the correct delivery of packets across the network. Additionally, the last segment of the IP address represents the node ID, which plays a key role in the organization and synchronization of the TDMA round. The Figure 3.3 shows the transmission of a packet in a multi-hop network. the topology is a line formation between 1 - 2 - 3. The node 1 wants to send a packet to node 3, which is accessible through node 2.

To ensure that the IP addresses remain intact during packet transmission, we perform the routing at layer 2 - The data link layer. Routing at the MAC layer involves managing communication between devices based on their MAC addresses rather than manipulating IP addresses. By routing

Figure 3.3 - Flow of the transmission of a packet from node 1, through node 2, to node 3

at this lower layer, the IP header, which contains the source and destination IPs, is left unchanged. The traffic can be directed efficiently across the network based on the nodes' physical addresses.

Following the example of Figure 3.3, for node 1 to send packets to node 3, it must first know that node 3 is reachable via node 2 and uses the MAC address associated with node 2. In this process, node 1 sends its packet to node 2, which acts as a router. Upon receiving the packet, node 2 consults its local routing information to forward the packet to the appropriate destination. In this case, node 2 knows that it can communicate with node 3 and forwards the packet to the MAC address corresponding to node 3. The Figure 3.4 represents this example.

Figure 3.4 - Flow of the transmission of a packet from node 1, through node 2, to node 3. Node 1 sends the packet destined to node 3 with the MAC address of node 2. Node 2 acts as a router and redirects the packet to its known destination, corresponding to the MAC address of node 3, the final destination. Node 3 receives the packet as it was sent by node 1.

When node 3 receives the packet from node 2, the packet still carries the source identification of node 1. This means the application running on node 3 remains unaware of the intermediate routing process that occurred at the MAC layer (Layer 2 of the protocol stack). The packet appears to have come directly from node 1 without indicating that node 2 acted as a relay. The goal of this algorithm is to enable seamless communication between nodes without requiring applications to be aware of the dynamic routing that occurs during transmission. As routes can change during the packet's journey, this Layer 2 routing ensures that the IPs remain unchanged, allowing any application to function as if operating in a direct peer-to-peer environment, even in a multi-hop network.

3.2.1 Routing Matrix

The entire routing algorithm is implemented within the routing module using C code. To implement the routing matrix concept outlined in the Section 3.1.2, the decision was made to convert

the routing matrix into a linked list structure. A linked list provides a more flexible and dynamic way to add, edit, or remove elements compared to other data structures. Given that the network topology is dynamic, the routing matrix can change frequently, making a dynamic data structure like a linked list an ideal choice. It allows for efficient updates without requiring continuous resizing, as would be required with arrays. While a doubly linked list offers additional functionality by allowing traversal in both directions, it introduces unnecessary complexity and overhead. For the purposes of this routing algorithm, a singly linked list is sufficient, offering the right balance between simplicity and flexibility without the additional cost of maintaining backward links.

Following the visual representation of the routing matrix in Figure 3.2, each matrix has two columns, one containing the direct links, and the other has the corresponding reachable nodes. Since the set of reachable nodes can change dynamically, a linked list is used to store the corresponding nodes, too. Therefore, in this approach, there is a primary linked list that contains nodes with direct links to the source node. Each primary node also has its own (secondary) linked list, which stores the nodes that can be reached through them. If a node has no further reachable connections, its list remains empty. The Figure 3.5 illustrates this concept applied to the routing matrices of Figure 3.2.

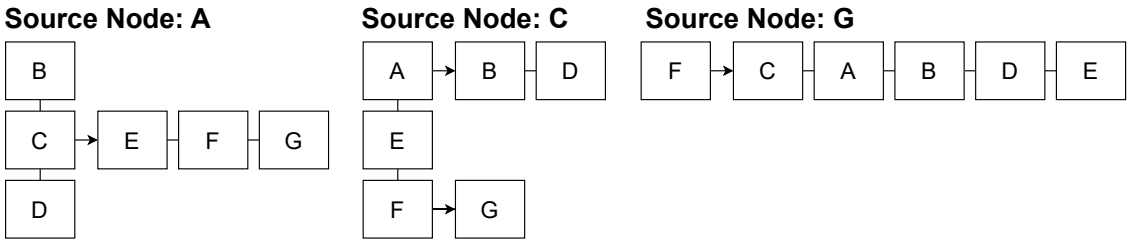


Figure 3.5 - Linked lists that store routing matrix information for source node, A, C, and G, respectively

As previously mentioned, when a node cannot communicate directly with another node, it forwards the packet to the MAC address of a reachable node from its primary linked list. This intermediate node, in turn, has the destination node in its linked list. As a result, the source node must know the MAC address of each node in its primary linked list. With this information, it can determine where to forward the packet, ensuring it reaches the correct final destination. Section 3.2.3 explains how MAC addresses are obtained and stored for later use in the routing algorithm.

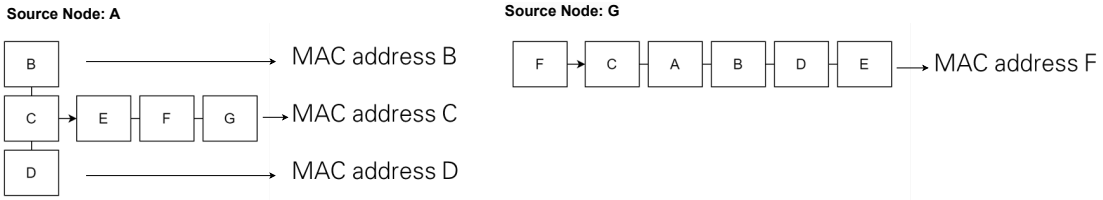


Figure 3.6 - For each node in the primary linked list, its linked list has its corresponding MAC address.

As shown in Figure 3.6, The MAC address of the directly connected node is stored alongside

the secondary linked list, indicating which MAC address should be used to forward packets to the reachable nodes. For the case where the source node (like G) only has one direct link (F), it automatically concluded that all the nodes are reachable through it (F).

Once the routes are established, the routing matrix is constructed, and the paths are defined, the next step is implementing this concept. In this system, routing occurs at the data link layer (Layer 2), bypassing traditional Layer 3 routing methods. One of the key objectives is to use as many of the existing software components and resources available on the devices as possible, particularly within the Linux/Raspberry Pi environment. This ensures the method remains easily applicable to similar devices running the same software, with minimal intervention in the device's existing functionality.

For a node to route a packet correctly, it must have the necessary destination information stored in its address resolution table, such as the ARP table, which maps IP addresses to MAC addresses. This table is essential for identifying the correct MAC address associated with the destination IP address, enabling the node to forward the packet to the appropriate next-hop node on its path to the final destination. By using the ARP mechanisms already present in Linux, the routing system can function efficiently without significant modification.

This approach simplifies implementation and enhances the system's portability, allowing it to be deployed across multiple devices without requiring major changes to the routing algorithm or device configuration. As a result, the system remains efficient, flexible, and easy to maintain in dynamic network environments.

By performing a system call to access the ARP table, the source node can update the IP addresses of each node in the network with the corresponding MAC addresses stored in its linked list. Once this mapping is added, it will be marked as "CM" (Complete Mapping), indicating that the IP-to-MAC address entry is valid and fully resolved. The device now has the corresponding MAC address for the given IP address, allowing it to communicate with the destination. The kernel uses these ARP entries to determine the appropriate MAC address for each destination IP and forwards packets accordingly. This allows the routing process to occur dynamically without requiring manual intervention, ensuring efficient and seamless packet transmission between nodes while maintaining IP-level communication.

The Figure 3.7 illustrates the process a packet goes through in the protocol stack. Node 1 sends a data packet to Node 4, with the path passing through two relay nodes: Node 3 and Node 2. Node 1 sends the packet with Node 3's MAC address. When Node 3 receives the packet, it verifies the matching MAC address. Upon reaching the Network layer, the kernel detects that the IP address does not match its own and re-routes the packet based on the information in the ARP table, forwarding it to Node 2. The packet goes through the same process at Node 2, which forwards it to Node 4, the final destination. Only at the source (Node 1) and the destination (Node 4) do the packets traverse all layers of the OSI model, reaching the Application layer.

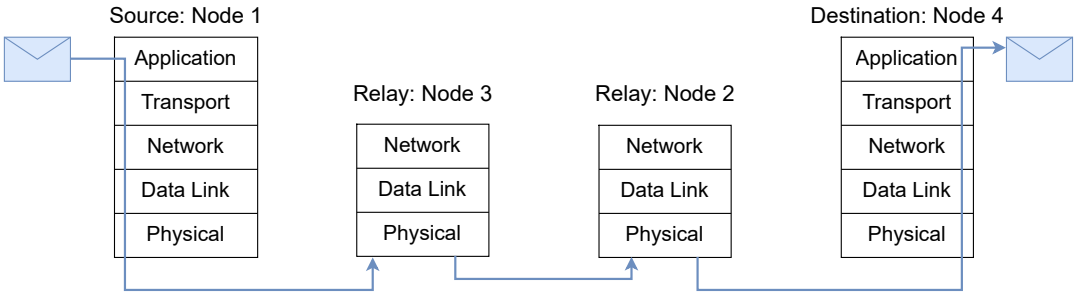


Figure 3.7 - A multi-hop communication process. The data packet is transmitted from Node 1 through two relay nodes (Node 3 and Node 2) before reaching its final destination at Node 4. At each relay node, the packet traverses through the lower layers of the OSI model (Physical, Data Link, and Network layers)

3.2.2 Configurations

For a node to function as a router, a specific set of configurations is essential for its proper operation. We change permanently the "$/etc/sysctl.conf$" file. This file is used to modify kernel parameters at runtime on Linux systems. Specifically, the chosen settings adjust various network behaviors for IPv4. To apply the changes, we must run *sudo sysctl -p* after adding the configurations to the $sysctl.conf$ file

- **net.ipv4.ip_forward = 1**: This enables IP forwarding, which is necessary for the device to route packets between interfaces;
- **net.ipv4.conf.all.accept_redirects = 1**: This tells the system to accept ICMP redirects on all interfaces. In a controlled network, accepting redirects is often disabled for security reasons.;
- **net.ipv4.conf.default.accept_redirects = 1**: This sets the default for new interfaces created after boot to accept ICMP redirects;
- **net.ipv4.conf.all.send_redirects = 0**: This disables sending ICMP redirects from all interfaces;
- **net.ipv4.conf.default.send_redirects = 0**: This disables sending ICMP redirects by default for any new interfaces.

These settings are essential to ensure that the node can function as a router, allowing it to forward packets that are not destined for itself rather than discarding them. This also enables the proper exchange of packets between interfaces such as the ad hoc interface, $wlan0$, and the tun interface, $tun0$. Additionally, these configurations ensure that when using system utilities like ping, nodes acting as relays can forward the packet to its correct destination instead of responding with an error message or a redirect. This is crucial for enabling seamless communication and routing in multi-interface and multi-hop network setups.

3.2.3 TDMA framework adaptations

The information about the network is already shared in the Shared Memory, with the structure of *spanningTreeSHM_t*. It is necessary to add a way to share and save the MAC addresses of the neighboring nodes.

It was decided to include the MAC address in each node's status share. Now, each node transmits its MAC address as part of its state within the synchronization packet. This addition introduces a small header of 6 bytes, which is considered negligible compared to the overall amount of data exchanged during each round. Importantly, this approach ensures that the MAC address is only shared with nodes that are in direct communication, limiting access to those that need it. Furthermore, the MAC address is only retained for as long as a connection exists, ensuring privacy and efficient use of resources. The state is updated from Figure 2.9 to the Figure 3.8. The structure is also updated for the Code Block 3.1, and the serialize and deserialize functions are corrected to this new structured form.

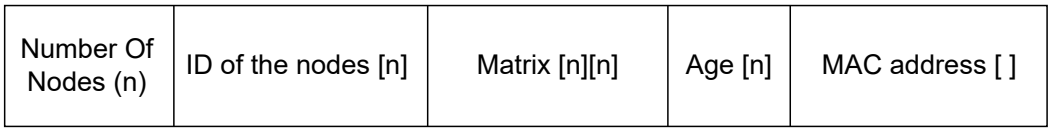


Figure 3.8 - The new state share now adds the bytes of the MAC address

Each node keeps an internal table mapping MAC addresses to node IDs, *macArraySHM_t*, that stores the MAC addresses of connected nodes as long as the connections are active. When the source node removes a link, the MAC address of the corresponding node is also deleted from the array, ensuring that only active connections are tracked. This structure is added to the Shared Memory.

```
1 typedef struct{
2 uint8_t myId;
3 uint8_t macAddress[MAC_BYTES];
4 uint8_t numberOfActiveNodes;
5 uint8_t matrix[MNUM_DRONES][MNUM_DRONES];
6 uint8_t idOfActiveNodes[MNUM_DRONES];
7 double  creationTime[MNUM_DRONES];
8 double  age[MNUM_DRONES];
9 } __attribute__ ((__packed__)) tdma_matrix_t ;
```

Code Block 3.1 - Update of the tdma_matrix_t structure

The structure presented in Code Block 3.2 is used to store complete information about each MAC address. It includes three key components:

- The **current size** of the array, which tracks how many MAC addresses are stored.
- A **mutex**, ensuring thread-safe access to the array when it is being modified or read.

- The actual **array of type `macInfo_t`**, which stores the detailed information.

Each entry in the *macInfo_t* array contains the **MAC address** (stored in bytes) and the corresponding **node ID**, which is represented by the last segment of the node's IP address. This structure allows for efficient tracking and management of connected nodes and their corresponding MAC addresses.

```
1  typedef struct {
2  uint8_t id[4];
3  uint8_t mac_bytes[MAC_BYTES];
4  } macInfo_t;
5
6  typedef struct {
7  pthread_mutex_t mut_macArray;
8  macInfo_t array[MNUM_DRONES];
9  size_t size;
10 } macArraySHM_t;
```

Code Block 3.2 - Structure of the array that saves the MAC addresses

The structure used for the linked list is in Code Block 3.3.

```
1 typedef struct routingList_ {
2 uint8_t id;
3 uint8_t macAddress[MAC_BYTES];
4 struct routingList_* next;
5 struct routingList_* accessibleNodes;
6 } routingList_t;
```

Code Block 3.3 - Structure of the linked list that saved the routing matrix information

3.2.4 Algorithm for routing table updates

The Figure 3.9 shows an example of a topology change, resulting in the presented Spanning Tree. Node H enters the network, and node E changes its position. Therefore, each node in the network now needs to update its routing matrix. The Figure 3.10 shows the routine matrix before and after the update. Node A can access the new node, H, through C. Node E, even tho it changed positions, in the perspective of node A, is still accessible through C, which imposes that there should not be a change in the matrix.

To ensure quick and efficient updates to the linked lists —specifically by avoiding unnecessary updates to data that do not affect the final view of the routing table — a small algorithm was developed. This algorithm identifies only the changes that have a direct impact on the routing table foundation, triggering updates only when necessary. This selective approach minimizes redundant processing and ensures that the ARP table is updated only when changes in the routing

information genuinely require it, optimizing both time and resource usage in maintaining network stability.

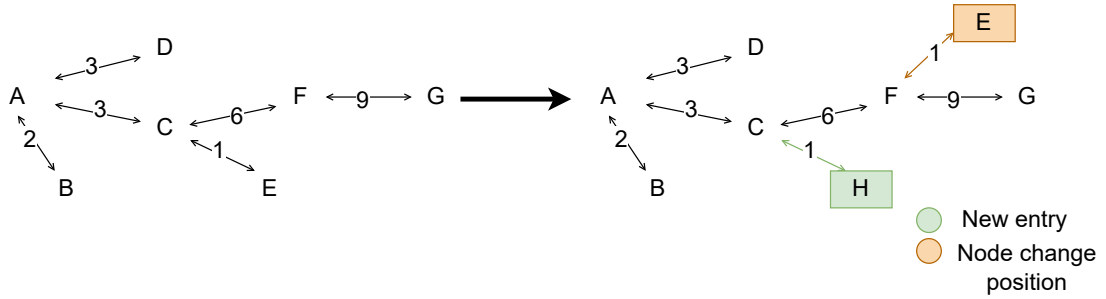


Figure 3.9 - Example of a topology change in the Spanning Tree, where node H enters the network, and node E changes position

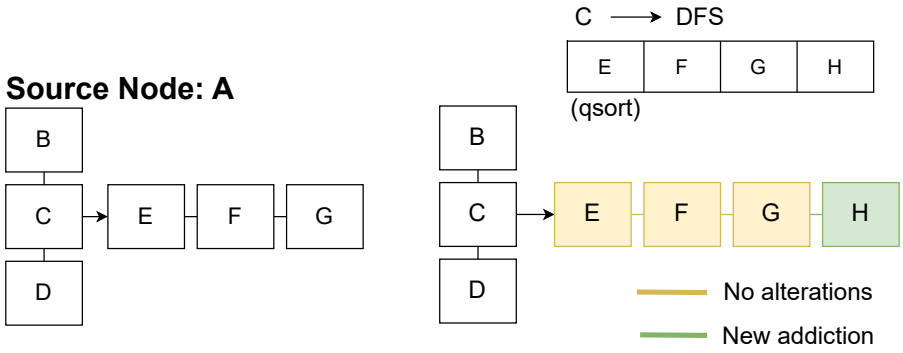


Figure 3.10 - The linked list is updated in an efficient way for the source A routing linked list

The Figure 3.10 illustrates the application of the algorithm for this case. First, we must verify if any of the nodes of the primary linked list has changed. If there is a change to this main list, we need to update all the nodes that were adjacent to the MAC address of that position because they are no longer accessible through that node. The nodes from the main linked list correspond to the ones that have a direct connection to the local node. The information for these connections is retrieved from the spanning tree. To obtain all the connections, we perform a Depth-First Search (DFS) starting from the local node. This DFS explores all reachable nodes, traversing as deeply as possible along each branch before backtracking. Once we have traversed all connections, we save the information in a sorted manner using *qsort*. This is important, as it allows us to easily compare and identify any changes in the connections over time. First, we update the main list, and then we update the secondary lists of each node in the main list.

For this example, we would verify that the main list is still the same, which means that the used MAC addresses are the same. Then, we perform the same procedure on each node, and we identify a direct connection. B and D would return that they don't have accessible nodes, as before.

For the node C, we receive an array (after *qsort*) as: [E, F, G, H]. The algorithm will compare the linked list of node C with the array. In the linked list, each element has the ID, so we can see

if there is a match. These are the situations possible by comparing the ID (which are numbers in the framework) from the linked list to the ID in the array:

- **They are equal:** This means that this ID was already in the routing matrix and is updated in the ARP table. So we advance to the next element in the linked list and in the array;
- **The linked list ID is greater than the ID in the array:** This means that the ID from the array is not updated in the routing matrix, so we should add it to the linked list and update the ARP table. Here, we advance to the next value in the linked list;
- **The linked list ID is smaller than the ID in the array:** This means that the ID from the linked list changed positions and is now not accessible from this main node. Here, we advance to the next value in the array;

If we reach the end of the linked list, we add any ID left in the array. If we reach the end of the array first, we must remove any left ID from the linked list.

Following the example, we first check if all values from the C linked list are present in the array. We identify that H is the only node missing. Therefore, we add H to the linked list, and at the end, we will perform an ARP add operation using H IP and C MAC address to update the ARP table.

This algorithm efficiently updates the network information by focusing only on relevant changes. In this case, the information from other nodes, like E, which changed position in the topology, does not require modification. From A perspective, E position remains unchanged, so there is no need to alter any data related to it. This approach optimizes the update process, ensuring that unnecessary changes are avoided, minimizing overhead, and maintaining accurate routing and ARP information.

In the end, we have the linked list updated with the view of the updated routing matrix. We saved which nodes need to be updated in the ARP table and their corresponding MAC addresses. We only need to delete a node from the ARP table when the members of the synchronization group change (when one leaves). When this situation is identified, we run an ARP delete to that IP entry. At the beginning of every routing actualization, we should verify if there is a change in the active IDs in the group to remove obsolete information from the ARP table. The linked list will be automatically updated to remove the obsolete node in the next iteration.

The next example shows what happens when there is an entry and a remotion of nodes in the main linked list. For this example, we used the C point of view in the network, 3.11. We perform the same base algorithm for any of the linked lists (main or secondary). When running the DFS for the local node, C, we identify that E is not in the results, and we have a new node, H. We add H and remove every information related to E. If E had nodes in its linked list, when performing DFS in the rest of the nodes, we would update the IDs to the new correct location. When performing the DFS in F and then comparing the results to the linked list, we keep the node G and add it before the node E. In the end, the ARP table receives two updates, node H and node E.

The routing algorithm is capable of rapidly updating changes, either immediately or shortly after transmissions within the slots begin. However, if there is a large number of nodes, the update

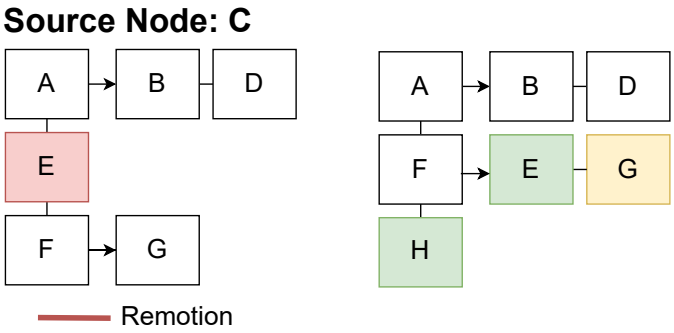


Figure 3.11 - The linked list is updated in an efficient way for the source C routing linked list. For this case, H is added to the main list. E is removed and later added to the F linked list. H and E will have updates in the ARP table

process will naturally take longer due to the increased complexity. In our tests, we confirmed that the algorithm remains fast and efficient, even with up to 10 nodes. This demonstrates that the algorithm can handle moderate network sizes while maintaining timely updates to the routing table, ensuring the network remains synchronized and responsive.

The main concern is how to efficiently execute ARP commands. The functions $popen()$ and $system()$ take nearly an entire slot duration to add ARP entries for three nodes, which is significant. If a new node joins the network, this delay would result in the loss of an entire slot duration just for the routing update, potentially disrupting ongoing communications.

By using $ioctl()$ (input/output control), however, we can send all ARP updates in a single operation, enabling the ARP table to be updated in a negligible amount of time compared to other options. This ensures minimal disruption to network operations.

$ioctl()$ is a system call in Unix-like operating systems that allows a program to directly communicate with or control a device driver or kernel-level functionality. It provides a programmatic interface to modify the ARP table efficiently, making it far faster and more suitable for real-time network environments than $popen()$ or $system()$.

Finally, the routing module will function as presented in the Figure 3.12.

3.2.5 Using TunTap for Standard Interfaces

In the TDMA framework, TUN/TAP is utilized to capture packets from the network interface, allowing them to be processed by the framework. Initially, the design allowed applications to send packets directly to the *tun*0 interface. This interface operates on channel 3 and is assigned an IP address in the 192.168.3.X range, where the last segment corresponds to the node ID. However, this implementation does not work to enable communication between two nodes on the network via routing.

The applications must send the packets to a node in the ad-hoc interface (*wlan*0), which has a 192.168.2.X IP range (X is the node ID). Consequently, it is necessary to route packets from the

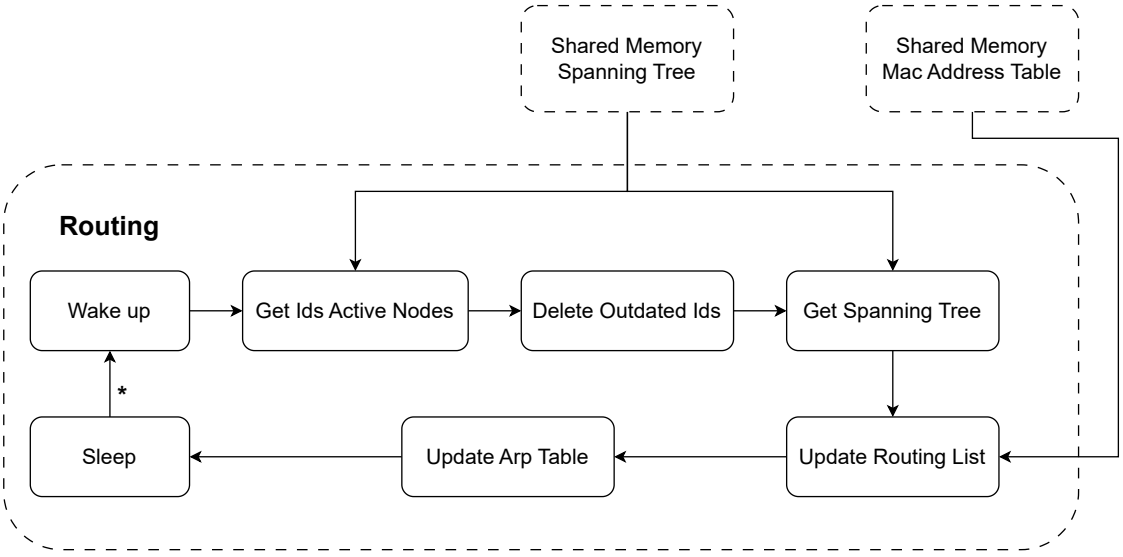


Figure 3.12 - Stages of the Routing module operation

*wlan*0 to *tun*0 while preserving the integrity of the packet headers. This ensures that the applications can seamlessly communicate across the network without being affected by the underlying routing or framework mechanisms.

To ensure packets are transferred from one interface to another without altering their contents or introducing additional delays, we used *iptables* rules. We need to route the *wlan*0 packets to the *tun*0; however, not all packets should be routed through the TDMA framework. Only packets originating from the applications that are leaving the network need to be sent to the TDMA framework. This distinction is crucial because the TDMA framework sends its own packets over UDP via the *wlan*0 interface. To avoid creating a loop, ensuring that these TDMA-generated packets are not mistakenly routed back to *tun*0 is essential. Proper handling of routing rules is critical to prevent such loops and maintain efficient network operation.

The mangle table in *iptables* is used to modify or mark packets as they pass through the network stack. It can be used to mark packets for special processing (like ensuring they are sent via a particular interface). We can use this tool to mark all the *wlan*0 packets that are locally generated, except the TDMA packets. By utilizing the OUTPUT chain in *iptables*, we can mark packets before they are sent out. To ensure that TDMA packets are excluded, we reset the mark to zero for packets belonging to the TDMA framework. Since all TDMA packets are sent via a UDP socket, they can be easily identified by their specific UDP port number, which follows a predictable pattern: 4000 + node ID. For example, on node 10, TDMA packets will use UDP port 4010. This marking mechanism ensures that only the relevant application packets are routed through the TDMA framework while excluding the framework's own UDP traffic, preventing unnecessary routing loops.

```
sudo iptables -t mangle -A OUTPUT -o wlan0 -j MARK --set-mark 1
sudo iptables -t mangle -A OUTPUT -o wlan0 -p udp --sport 40010 -j MARK --set-mark
    0

# Add a rule to route packets with mark 1 using table 100
sudo ip rule add fwmark 1 table 100

# Add a default route in table 100 to route packets through tun0
sudo ip route add default dev tun0 table 100
```

Code Block 3.4 - Bash script to run the *iptables* mangle configuration to route outgoing *wlan*0 packet to *tun*0 for node 10

When the *tun*0 is configured, we run a bash script to set the desired *iptables* configurations (Code Block 3.4).

When the TDMA framework receives a packet, it writes it to the *tun*0 interface. However, this approach presents a challenge: the packet cannot be directly sent back to the application, which resides on a different interface.

This is problematic because the framework processes the packet through *tun*0, but there needs to be a mechanism to forward it back to the original application without interfering with the intended routing or network setup. Simply writing to *tun*0 does not automatically return the packet to the application on another interface, which requires further routing and handling.

To send the packet back to the original interface, we need to use raw sockets. Raw sockets provide low-level access to network protocols, allowing us to send packets directly, bypassing the usual transport layer protocols like TCP and UDP. When using these types of sockets, we must run our programs with root privileges since they provide direct access to the network stack.

The *tun*0 interface only deals with IP packets; therefore, we must use the IPPROTO_RAW socket type: $sockfd = socket(AF\_INET, SOCK\_RAW, IPPROTO\_RAW)$;. In this way, the packet is ready to be sent. We do not need or want to change any header of the packet.

Finally, although the destination of the raw socket is *wlan*0, binding the socket directly to the outgoing interface causes the raw socket to bypass the kernel's processing. This means that the packet will be sent without the kernel handling fragmentation or reassembly. To address this issue, instead of binding the socket to the interface, we should bind it to the destination address of the packet. This can be done by retrieving the packet's destination IP. By binding the socket in this way, the kernel will process the packet, handling necessary tasks such as fragment reassembly and ensuring proper routing.

In conclusion, when the TDMA framework forwards a packet to tun0, if the packet is intended for the local node, it will be sent through the wlan0 interface, as outlined above. This method ensures that the kernel manages the packet processing, including reassembly, when necessary.

With this framework, it is now possible to use any type of application with TDMA framework with any packet size.

The Figure 3.13 demonstrates how a packet travels through the network from an application to the TDMA framework until reaching its final destination.

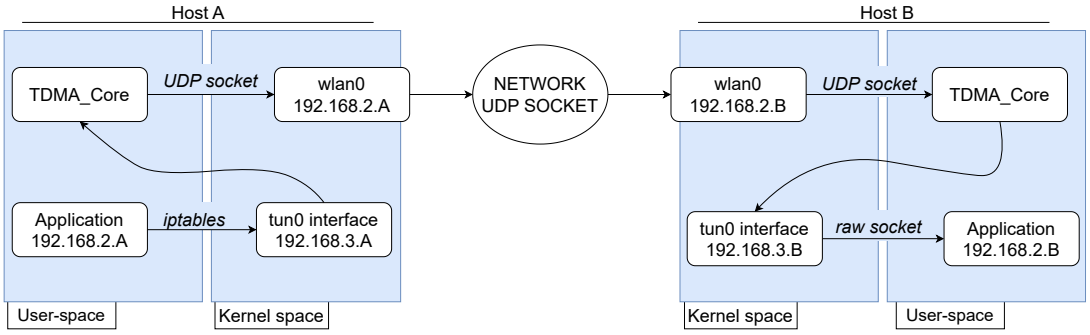


Figure 3.13 - Flow of packets in the framework TDMA from source (Host A) to destination (Host B).

3.2.6 Final TDMA framework

After the modifications, the final TDMA framework (fig. 3.14) will include a new module, the Routing module, which executes the described algorithm using the MAC address table and the Spanning Tree, both accessible via Shared Memory. Also, the Routing module updates the routes in the ARP table via system calls.

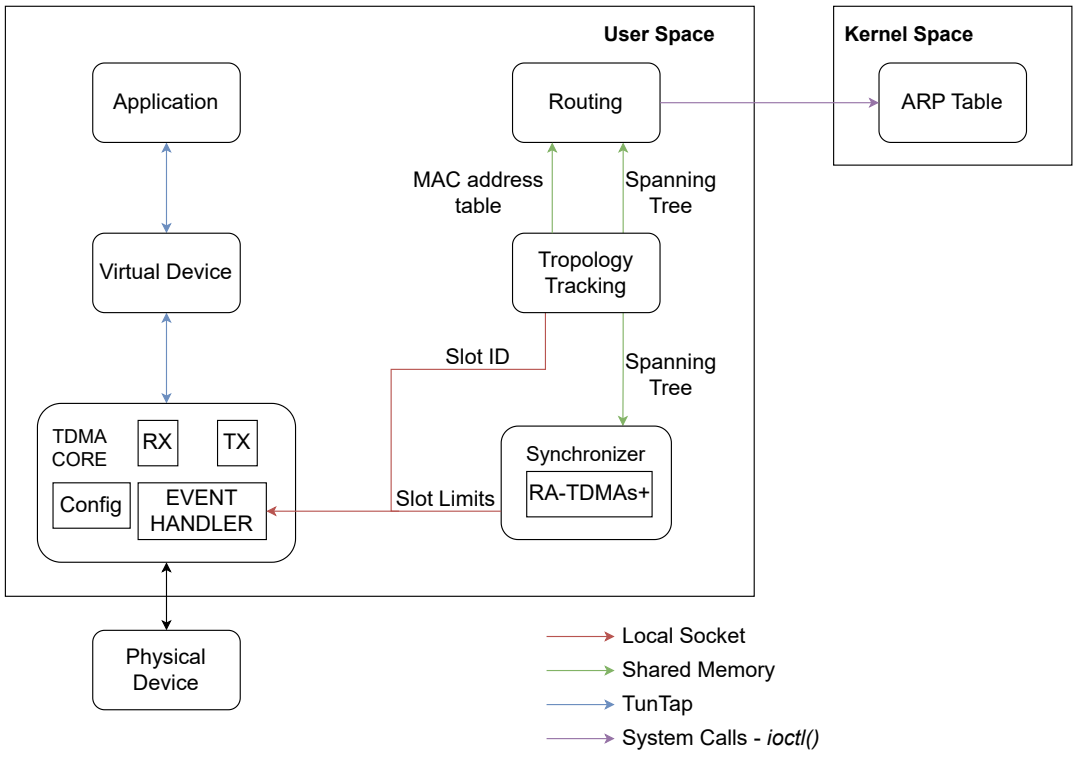


Figure 3.14 - Updated TDMA framework

In a multi-hop topology, packets are forwarded by intermediate nodes to reach their destination. When a node transmits a packet during its assigned TDMA time slot, the intermediate nodes forward the packet within the same slot, effectively acting as relays. This process is automatic and operates on behalf of the source node, ensuring that the packet is forwarded seamlessly across multiple hops. This type of automatic rerouting was proposed in [3], but it was validated only through simulation. However, if a packet is transmitted near the end of the source node's slot, the forwarding process may not be completed in time before the slot ends. As the packet traverses the intermediate nodes, the final destination may receive it late and out of sync. This can lead to interference and de-synchronization between nodes, disrupting the overall network timing.

To mitigate this issue, it would be beneficial to introduce a small transmission delay. This delay would prevent new packets from being sent too close to the end of the time slot, allowing sufficient time for each packet to propagate through the network and reach its destination before the slot expires. In this way, the system ensures that no subsequent packets are transmitted prematurely, avoiding rescheduling conflicts at the end of the slot.

When a node receives a packet that has been relayed by intermediate nodes, it interprets the information as though it was sent directly from the original source node. This means that the packet's data remains unaffected by the relay process, and there is no alteration or interference in the RA-TDMAs+ synchronization delays calculation. As illustrated in the Figure 3.13, the packets only enter the TDMA application at the destination node, ensuring that the relay process remains transparent to the final receiver. Even if the packet passes through multiple relays, the destination node will always perceive the packet as originating directly from the source node, maintaining the integrity and timing of the transmission.

This concept presents an opportunity for future work. A potential solution would involve developing an algorithm that dynamically calculates and applies a transmission delay based on factors such as the packet size and the number of hops required to reach the destination. This algorithm would ensure that packets are transmitted with enough time to complete their journey across the network within the designated slot, preserving synchronization and minimizing interference across the network.

3.3 Summary

This chapter provided a detailed explanation of both the design and implementation of the routing algorithm. The algorithm was developed to ensure optimal performance within the TDMA framework. By utilizing topology information from the RA-TDMAs+ protocol, routes can be determined and implemented using standard mechanisms of the Linux operating system and its network stack. This approach allows the framework to operate without modifying or adding to the existing system, ensuring minimal intervention in the nodes current functionality. This way, it is possible to maintain communications between any two nodes in the network, enabling the exchange of information transparently without the need for applications to be aware of the underlying routing and synchronization structure. The routing technique operates at the data link

layer, utilizing other network nodes as routers responsible for relaying packets. By utilizing the information from the connectivity matrix to know the topology, the dissemination of this matrix throughout the network enables the routes to converge on a unified view among every node in the network.

Chapter 4

Experiments and Results

This chapter provides a comprehensive overview of the experimental setup and the metrics deemed relevant for assessing the performance of the network implementation. It begins with a discussion of the experimental setup, including the equipment utilized across various experiments. Following this, a brief description of the AlphaBot2-Pi, the mobile node employed in the tests, is presented.

The chapter then outlines the specific experiments conducted to validate the network implementation, detailing the methodology used and the metrics that will be employed to evaluate the results. In each experiment the calculation methods are explained, along with the significance of its outcomes. The results of the experiments are subsequently presented, accompanied by an analysis aimed at drawing meaningful conclusions. The chapter concludes with a summary of the key findings and their implications for overall network performance.

4.1 Experimental Setup

The testbed consists of four wireless ad hoc network nodes. The ad hoc network runs in the interface *wlan*0. The nodes have static IP addresses in the 192.168.2.X range, where the last segment of the IP corresponds to the node ID. From the perspective of the IP layer, all nodes are considered to be part of the same network. All network cards support ad-hoc mode. The ad hoc network has a Maximum transmission unit (MTU) of 1500 bytes.

Wireless communication in the network operates on various channels, with channels 1, 6, and 11 being the preferred options since they do not overlap, reducing potential interference. In an infrastructured network, the communication channel can be dynamically adjusted based on performance requirements. However, in an ad-hoc network, the channel is manually assigned and cannot be changed automatically. Considering that channel 6 is widely used and often congested, channel 1 was chosen for this ad-hoc setup to minimize interference and ensure a more stable connection.

The network was initially configured with a default retry value of 7. To assess potential improvements, the retry value was adjusted to 5 in order to determine if this lower value could reduce latency spikes caused by excessive retries. Although the network performed well with 5 retries,

especially in the short term, it was observed that over time, the higher retry value of 7 allowed TDMA to function more smoothly. While 5 retries showed good initial results, 7 retries provided better performance in the long run by compensating for time trade-offs during transmission, ensuring more reliable and consistent communication.

Devices Used:

- Two Raspberry Pi 4 Model B units, configured with the Raspberry Pi OS.
- One Raspberry Pi 4 Model B embedded in an AlphaBot2-Pi with an integrated camera.
- A computer running Ubuntu participates in the network and is central for monitoring the ad hoc network and overseeing packet transmission.

Table 4.1 - Network information of the active nodes

Model	IP	MAC Address
RPi 4 Model B 1.5	192.168.2.5	e4:5f:01:c0:7c:8b
Ubuntu PC	192.168.2.6	00:21:5d:1b:85:7c
RPi 4 Model B 1.4	192.168.2.7	e4:5f:01:6c:4d:ce
RPi 4 Model B 1.1	192.168.2.10	dc:a6:32:45:40:fa

The primary focus is on evaluating the routing performance. In the first phase, tests will be conducted without the TDMA framework, allowing for an analysis of how the developed algorithm functions in a multi-hop ad hoc network. In the second phase, tests will incorporate the TDMA framework with the RA-TDMAs+ synchronization algorithm, enabling a comprehensive assessment of routing performance within the structured TDMA communication environment. The round period is 100 ms, meaning that in a network of four nodes, each node has 25 ms of slot time. The Figure 4.1 shows a representation of the TDMA when the four nodes are active and synchronized.

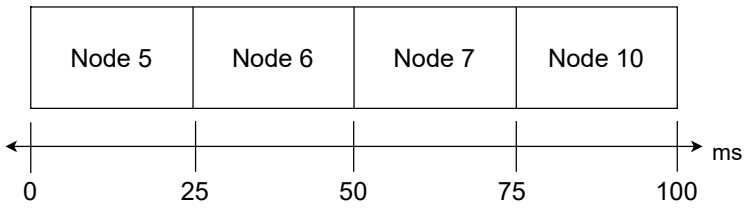


Figure 4.1 - Representation of the TDMA round when the four nodes run and are synchronized. Each slot has 25 ms.

We developed a small script that simulates the network topologies to test the routing algorithm independently of the TDMA framework. This script generates the corresponding Spanning Tree structures for the predefined test topologies. These Spanning Tree structures are then provided to the nodes, allowing the routing algorithm to execute as designed. In this way, we can evaluate the routing performance by simulating the generation of the necessary Spanning Trees without relying on the TDMA framework.

The simulated topologies are arranged linearly for testing purposes, ensuring that nodes are forced to use multi-hop routing to send packets across the network. This setup helps evaluate the routing algorithm's performance by requiring each node to relay data through intermediate nodes to reach the destination. The AlphaBot moves through the network, re-arranging the topologies and the routes. The Figure 4.2 shows two options of topology where node 10 and node 7 have direct communication (P2P), the Figure 4.3 represents the topology with a double hop communication, and the Figure 4.4 with a triple hop. The faded robot, node 7, represents the previous position of the robot. The nodes 5, 6, and 10 can not move, so the robot moves and makes the network a dynamic multi-hop network. We do this type of topology to impose routes specific routes for testing.

To perform tests during topology changes, two multi-hop network scenarios were defined:

- **Multi-hop Network 1:** The network is initially configured as a Point-to-Point (P2P) topology and then switched to a 3-hop configuration.
- **Multi-hop Network 2:** The network is first set up as a Point-to-Point (P2P) configuration, then changed to a 2-hop configuration, and finally switched to a 3-hop configuration.

These tests aim to evaluate the performance and behavior of the network during dynamic topology changes, focusing on the impact of re-routing and transmission delays introduced by adding intermediate relay nodes.

Although only one AlphaBot was used, the testbed effectively simulates the characteristics of a MANET, providing the essential conditions to evaluate the routing algorithm. This controlled environment ensures consistent and repeatable experiments, allowing us to draw reliable conclusions about the network's performance and the effectiveness of the routing algorithm across different scenarios.

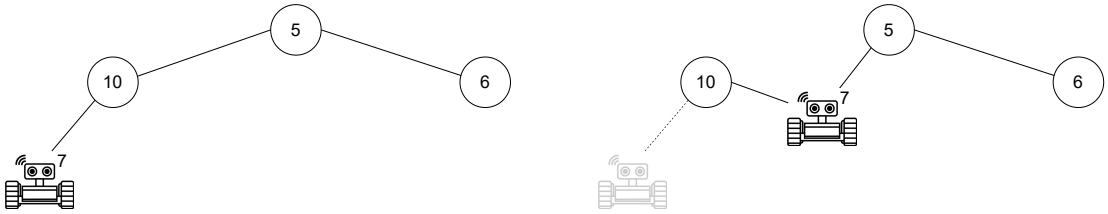


Figure 4.2 - Topologies for peer-to-peer communication between node 10 and 7

A network card was added to each node, enabling the use of wireless infrastructure for executing remote operations. This setup allows the nodes to be controlled remotely, simplifying code modifications and enabling real-time network monitoring with tools like Wireshark and TCPdump. Remote access is facilitated via SSH or VNC from an external computer. Additionally, the network card can operate in monitor mode, capturing probe, management, and data packets exchanged within the network. This way, it is possible to verify if the TDMA framework is working and the synchronization is being achieved. Also, by analyzing the radiotap header contained in

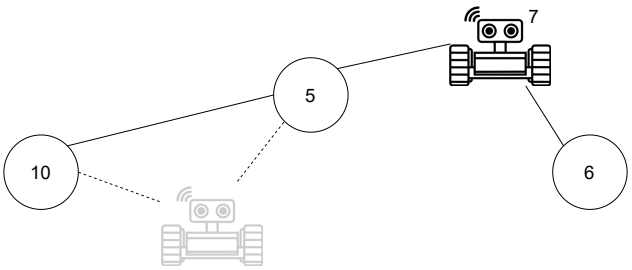


Figure 4.3 - Topology for double-hop communication between node 10 and 7

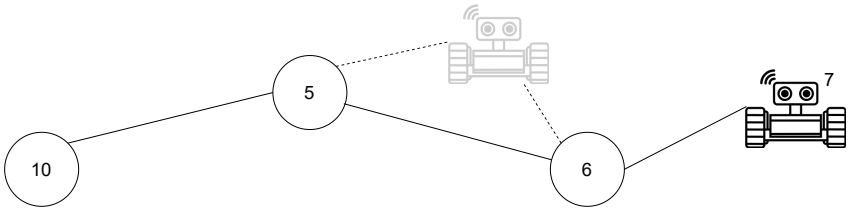


Figure 4.4 - Topology for triple-hop communication between node 10 and 7

each packet captured in the mode monitor, it is possible to assess the quality of the links to each node by extracting the Received Signal Strength Indicator (RSSI).

The devices were strategically positioned at varying distances to create realistic link quality variations. However, the range of the wireless ad hoc network precluded the creation of low-quality links or complete disconnections within the laboratory environment, and changing the power transmission was not possible. To overcome this limitation, the Linux firewall (*iptables*) was used to block packet reception from specific IP addresses, with the command $iptables - A\ INPUT - s < IP > - j\ DROP$, thereby enabling the simulation of diverse network topologies. This selective blocking facilitated the controlled testing of various connectivity scenarios and network conditions. Since the topologies must be manually forced using bash commands executed via SSH for each node, a small extra delay is introduced during the topology change due to the time required for manually setting up the topologies.

4.1.1 AlphaBot2-Pi

The AlphaBot2-Pi (figure 4.5) is a versatile robotic platform that integrates with the Raspberry Pi 4, offering a complete solution for various robotics applications. It is equipped with:

- **Two DC gear motors** that control the robot's wheels. These motors are independently controlled, enabling the robot to perform complex movements such as forward motion, turns, and rotations.
- **Infrared (IR) sensors** located at the front and underside of the robot. The front IR sensors detect obstacles, while the underside IR sensors are used for line-tracking, allowing the robot to follow predefined paths or avoid collisions.

- **Ultrasonic sensor**, which provides additional obstacle detection and distance measurement for more accurate navigation in dynamic environments.
- **Two Servo Motors** that control movements of the camera. It is possible to move the camera left and right, as well as tilt it.

In this setup, the **Raspberry Pi Camera Module** is used, typically mounted on the robot, enabling it to capture images or video for tasks such as real-time video streaming, object detection, and image recognition. The camera is compatible with the Raspberry Pi 4, providing up to 8MP resolution for detailed image capture.

The IR sensors positioned at the front of the robot were used to detect nearby objects and obstacles in its path. By continuously monitoring the distance to obstacles, the robot could adjust its trajectory to avoid collisions autonomously. The real-time feedback from these sensors allowed the robot to make quick decisions about stopping, turning, or rerouting when an obstacle was detected.

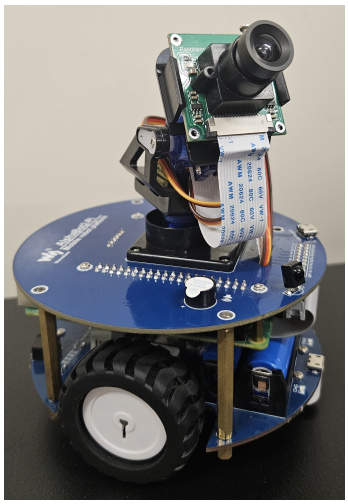


Figure 4.5 - AlphaBot2-Pi used

4.2 Metrics

This section outlines the metrics that will be employed to numerically evaluate the system's performance. These metrics will be used to assess the framework's performance in peer-to-peer communications under a multi-hop network. We also developed a Python script to calculate the metrics from the stored logs, presenting the results through graphs and calculations.

These metrics will be used to assess the framework's effectiveness

4.2.1 Round-trip Time (RTT)

Round-trip time (RTT) is the elapsed time, measured in milliseconds (ms), required for a network request to travel from its source to its destination and then return to the source. This depends on

the distance between the two points and the number of hops between them. It encompasses various delays, including propagation, transmission, processing, and queuing delays, reflecting the latency in the network. RTT is essential for evaluating network performance; lower RTT values indicate a faster, more responsive network, which is critical for applications like real-time video streaming.

4.2.2 Packet Loss Ratio (PLR)

Packet Loss Ratio (PLR) is a metric in networking that quantifies the fraction of data packets that are lost during transmission from the source to the destination. It is the ratio of lost packets to the total number of transmitted packets as in Eq. 4.1. A high PLR indicates poor network performance due to significant packet loss, which can lead to issues like delayed or corrupted data.

$$PLR = \frac{PacketsTransmitted - PacketsReceived}{PacketsTransmitted} * 100\% \tag{4.1}$$

4.2.3 RTT Jitter

RTT jitter refers to the variability in the time it takes for data packets to travel from the source to the destination across a network and back. It measures the inconsistency in the network delay, which can significantly impact the quality of real-time applications. High jitter can cause delays, buffering, and poor user experiences. In particular, we use the average relative jitter as expressed in Eq. 4.2, which is the average between the differences of all consecutive N measurements of the respective RTT for a given pair of nodes.

$$Jitter = \frac{\sum_{i=1}^{n-1} |RTT_{i+1} - RTT_i|}{n-1} \tag{4.2}$$

4.2.4 Throughput

Throughput refers to the rate at which data is successfully transferred from one location to another over a given period. Throughput represents the actual data transfer rate experienced by users or applications, distinguishing it from theoretical maximum speeds or bandwidth. While bandwidth indicates the total capacity of a network link, throughput accounts for various real-world factors that can affect data transmission. These factors include network congestion, packet loss, latency, and protocol overhead, which can reduce the effective data transfer rate. Here, we measure throughput as given by Eq. 4.3, dividing the total data transferred in a given interval by the duration of that interval.

$$Throughput = \frac{TotalDataTransferred}{TotalTimeTaken} \tag{4.3}$$

4.3 Experiments

We conducted experiments to evaluate the network performance with and without the TDMA synchronization framework for comparative analysis. The tests included direct communication

and communication within a dynamic multi-hop network. The TDMA tests were performed once the network was fully synchronized and stable, with all nodes added, to focus on the efficacy of the routing mechanisms. Each test ran for at least two minutes to stabilize initial conditions and was repeated ten times. Data logging was performed using Wireshark or TCPdump.

Each TDMA round is 100 milliseconds long. Running the tests for 2 minutes equates to 1200 rounds per test. With 10 experiments, this allows us to observe and analyze 12,000 rounds. This extensive testing period enables us to thoroughly evaluate how communications adapt and synchronize when our routing algorithm enables peer-to-peer communication using different routes. By examining these rounds, we can gain insights into the performance and stability of the network under various conditions and routing scenarios.

We employed the command '$iperf3$' to simulate interferences to generate UDP packets to overload the links. The interference is defined by setting the bandwidth between 50-70% of the available throughput in the current configuration. This approach allowed us to create a range of interference scenarios, helping us evaluate the network robustness and performance under different levels of traffic load and interference.

We conducted a series of tests within the wireless network, beginning with peer-to-peer (1 hop) communication. Next, we introduced additional hops to the network and tested the system under a dynamic, changing topology.

4.3.1 Ping command

The initial test involves using the ping command to evaluate the network and understand how routing impacts communication flow. This command measures the Round-Trip Time (RTT) between a source and a destination. Increased congestion and higher hops between nodes result in higher RTT values. Consequently, a higher ping value indicates lower network performance. Each test runs for two minutes, being approximately 120 pings.

The test was performed with different sizes:

- 64 bytes: a normal small packet;
- 1408 bytes: a packet size near the common Data Link MTU used in several networks;
- 5008 bytes: a size that requires fragmentation and a reassembly process in the communication protocols stack. The ICMP packet request is fragmented into four packets in total, three with the size of the MTU and a last one with 602 bytes (including headers);

4.3.2 TCP application Client-Server

A TCP-based application was developed to assess the stability of connections within the network. TCP ensures reliable communication by retransmitting lost packets, though this comes at the cost of increased latency. In this application, packets are sent from the client to the server, with the client awaiting a response before sending the next packet. Despite the small payload size of each

packet (ranging from 100 to 500 bytes), messages are transmitted continuously for a duration of 2 minutes.

4.3.3 TCP connection via iperf3

The $iperf3$ tool can be used to infer specialized metrics about TCP performance, like throughput, bandwidth, and retransmissions. It measures the available bandwidth between a client and server, indicating how much data the network can handle. Additionally, $iperf3$ tracks TCP retransmissions, which occur when packets are lost and need to be resent, offering a view of network reliability. By analyzing these metrics, we can assess the efficiency and stability of TCP connections in more detail.

4.3.4 Video Stream

Lastly, we conducted a test involving a video stream where the AlphaBot transmitted the stream to a node using the TCP or UDP protocols. The *raspivid* tool captured the video stream from the Raspberry Pi Camera Module. The VLC application was used to access the video stream. The stream utilized the Real-Time Streaming Protocol (RTSP), which is a network protocol designed to control the transmission of multimedia streams over IP networks.

RTSP offers distinct advantages in a multi-hop ad-hoc network where latency, bandwidth, and packet loss are key concerns. It provides low-latency streaming, making it ideal for real-time video transmission across multiple network hops. Additionally, RTSP's ability to use both UDP and TCP for data transfer ensures better resilience to packet loss, which is common in such networks. This combination of low delay and robustness makes RTSP a strong choice for streaming video in environments where real-time performance and reliability are critical.

This test allowed us to evaluate the network performance under high packet loads. To ensure a stable stream under favorable conditions, we used a resolution of 320x240, a frame rate of 15 fps, and a bitrate of 300 kbps. Using TCP ensures a more stable connection, even if it results in slower performance than UDP, which tends to cause frequent frame breaks. Running tests for both cases helped compare the performances of each protocol. To perform the stream with TCP, we used the code present in Code Block 4.1. To use UDP, we simply remove the $proto = tcp$ segment.

When streaming video from *raspivid* (which can output video in H.264 format), we can instruct the VLC player to specifically expect an H.264 video stream and demultiplex it accordingly. The ": $demux = h264$" tells the VLC player to handle the stream as an H.264 video, which simplifies processing and playback by avoiding unnecessary detection or incorrect handling of the video codec.

The receiver then connects to the stream via VLC with the *"Open Network Stream"* option and plays the stream with a network URL, **"rtsp://192.168.2.7:8554/stream"**. This specifies:

- **rtsp://**: Specifies the RTSP protocol is being used;
- **192.168.2.7**: This is the IP address of the RTSP server (the Alphabot IP);

```
#!/bin/bash

# Define the stream resolution, frame rate, and bitrate
RESOLUTION="320x240"
FRAMERATE="15"
BITRATE="300000"

# Stream video using raspivid and VLC over TCP with RTSP
raspivid -o - -t 0 -w $RESOLUTION -h $RESOLUTION -fps $FRAMERATE -b $BITRATE | cvlc
     -vvv stream:///dev/stdin --sout '#rtp{sdp=rtsp://:8554/stream,proto=tcp}' :
    demux=h264
```

Code Block 4.1 - Command to start a video stream in the Raspberry Pi with TCP

- **8554**: This is the port number on which the RTSP server is listening. Port 8554 is a default port for RTSP, though it can be configured to use other ports.
- **/stream**: This is essentially a path or identifier that the RTSP server uses to differentiate between different streams.

The method used for video streaming sends frames in split packets, each containing up to 1400 bytes of data. This approach avoids kernel-level fragmentation and reassembly, reducing the overall transmission load and minimizing processing delays. As a result, the network experiences better performance with fewer interruptions during video streaming.

4.3.5 TDMA transmissions synchronization

We analyzed the nodes transmissions during TDMA synchronization. By using Wireshark and a node operating in monitor mode, we captured the timestamps of each communication. These timestamps can be plotted over time, allowing us to observe the periodicity of each node and the definition of time slots. This also provides insights into how the synchronization group is rearranged when a node joins or leaves the network and the packets traveling the multi-hop network.

4.3.6 Experiments Remarks

We conducted experiments transmitting data between node 7 and node 10, utilizing different multi-hop topologies under both normal conditions and with interference while varying packet sizes and network load. These experiments included direct communication, two-hop, and three-hop setups, as well as dynamic topologies where the network structure changed during the experiment. The objective was to evaluate the performance of the routing algorithm in updating routes and to assess the impact of using and not using the TDMA framework on communication synchronization. The experiments were evaluated in terms of delays, throughput, transmission variation, and the consistency of packet delivery. Additionally, we analyzed TDMA slot synchronization during packet exchanges in multi-hop environments to assess its effectiveness in maintaining orderly communications with the forwarding of packets through the relays in the multi-hop network.

It is important to note that, even among devices of the same model, internal variations exist. These hardware asymmetries can lead to differences in performance and behavior, contributing to variability in network performance. Among the Raspberry Pis used, node 10 was observed to have the poorest network performance compared to the other nodes. The remaining nodes were able to exchange packets more quickly with each other than when communicating with node 10.

4.4 Results Without TDMA

4.4.1 Ping Command Results

No packet loss was observed for every ping test, even under interference. This reliability can be attributed to the network configuration, specifically using 7 retransmission attempts before the transmission fails. In a wireless ad-hoc network, interference from external sources or obstacles can cause temporary disruptions in communication. However, the network overcame brief interruptions by attempting to retransmit the packet multiple times. Each time a packet fails to be acknowledged, the network automatically retries up to 7 times, significantly increasing the chances of successful packet delivery even in the presence of interference. This mechanism ensures robust communication, maintaining the integrity of data exchange despite potential signal disturbances.

The ping command has no view of packet retransmission attempts; however, the effect of these retransmissions is reflected in the increased RTT values. When retransmissions occur due to interference or packet loss, the additional attempts to deliver the packet result in a longer response time, which is observed as higher RTT values in the ping results. This indicates that while packets are eventually delivered, the network requires more time to ensure successful transmission under challenging conditions.

- **P2P (Direct communication - 1 hop)**

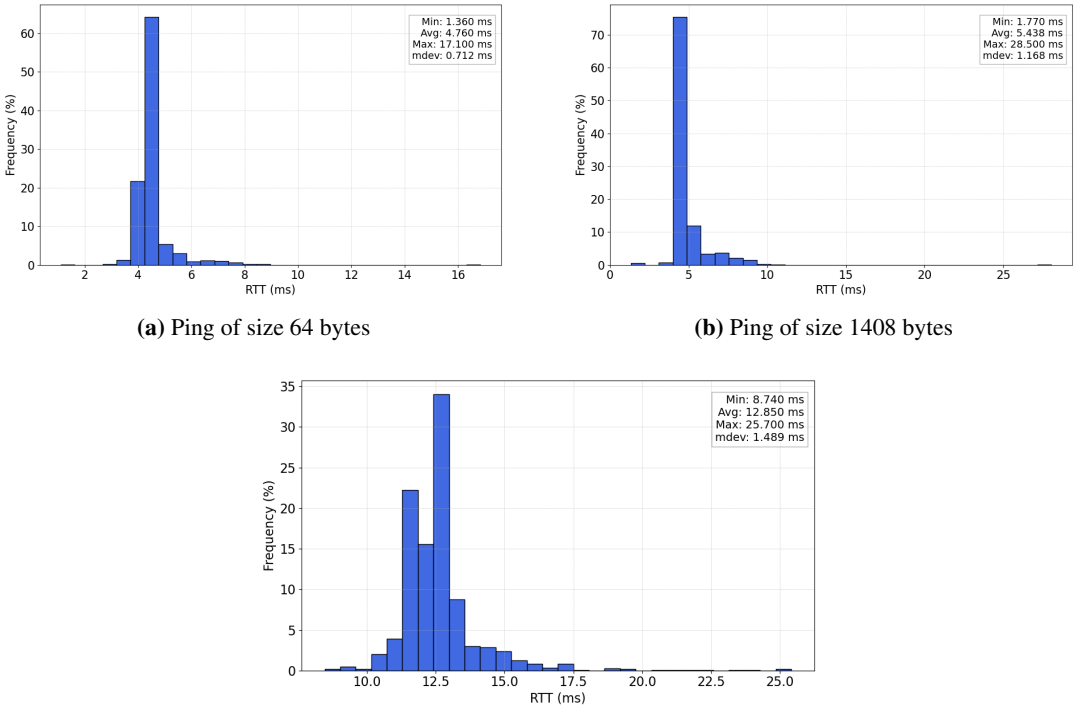
Small packet sizes in a P2P setup exhibit low latency and highly stable network performance (Figure 4.6a). The network handles small packets efficiently, with minimal overhead and variability, as these packets require less processing and are not subject to fragmentation or significant delays. This consistent performance is expected, given that smaller packets place less strain on the network infrastructure and can be transmitted more quickly and reliably.

As packet size increases, the RTT average latency and variability also rise (Figure 4.6b). This is typical in a P2P network, as larger packets require more processing time and are more susceptible to delays, particularly due to fragmentation and interference. Larger packets are more likely to experience network congestion or retransmissions, contributing to higher latency and variability.

For very large packets, the latency penalty becomes significant due to the overhead introduced by fragmentation and additional processing. The wider distribution of RTT values indicates that larger packets are more prone to network variability, congestion, and retransmissions, leading to inconsistent performance. The histogram (Figure 4.6c) shows that most packets cluster around 12.5 ms, reflecting some degree of consistent performance. However, the longer tails (with RTT

values reaching up to 25 ms) suggest that the network occasionally struggles, causing delays and performance degradation.

Fragmentation adds overhead and increases latency, but without interference, the network can still handle large packets with reasonable stability. However, this comes at the cost of reduced efficiency compared to smaller packets, as larger packets inherently demand more processing and are more susceptible to delays.



(a) Ping of size 64 bytes

(b) Ping of size 1408 bytes

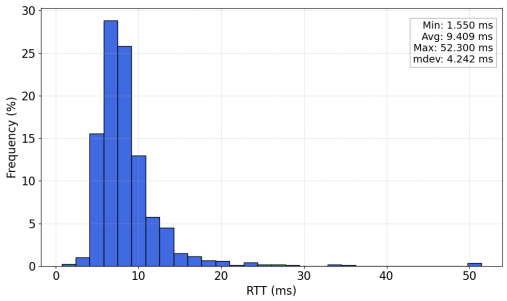
(c) Ping of size 5008 bytes

Figure 4.6 - RTT values of ping commands with different packet sizes in a P2P network

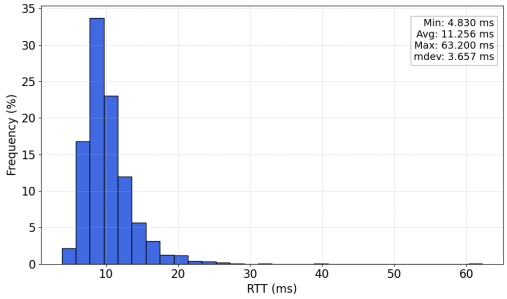
With interference, the graphics in Figure 4.7 show some outliers with high values. For 5008 bytes, 4.7c, there were several instances where RTT values exceeded 40 ms, with the highest recorded value reaching 103 ms. Both 3 cases show some variability in results.

The impact of retries is significantly more pronounced for medium and large packets. These packets take longer to transmit and are more susceptible to interference, increasing the likelihood of collisions with other transmissions. This, in turn, leads to more frequent retransmissions, further compounding delays. In the case of large packets, fragmentation intensifies the problem: each fragment must be transmitted successfully, and interference at any point in the sequence can trigger retransmissions of multiple fragments. This process adds substantial overhead, leading to increased RTT and higher variability, making the network performance less predictable, especially in high-interference environments.

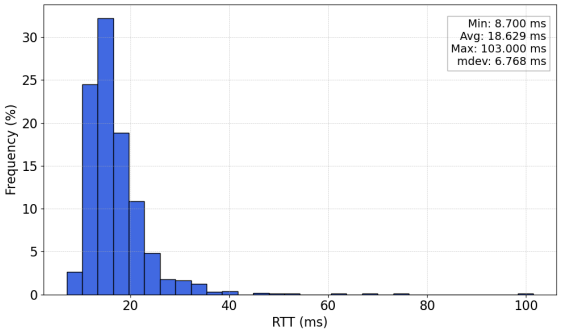
The results from these P2P tests provide a clear baseline for understanding how packet size impacts network performance. As expected, small packets (64 bytes) transmit efficiently with minimal delay and variability, while larger packets (5008 bytes) face considerable delays due to



(a) Ping of size 64 bytes



(b) Ping of size 1408 bytes



(c) Ping of size 5008 bytes

Figure 4.7 - RTT values of ping commands with different packet sizes in a P2P network with interference

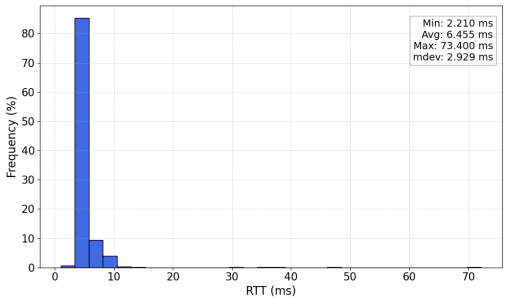
fragmentation. This pattern underscores how increasing packet sizes introduce greater overhead and make larger packets more susceptible to network limitations such as congestion and retransmissions.

These findings set the stage for evaluating multi-hop configurations, where RTT and jitter are likely to rise across all packet sizes. Large, fragmented packets are expected to experience the most significant delays, as each additional hop adds processing time and increases the likelihood of retransmissions and congestion. In contrast, small packets should maintain relatively stable performance. However, there may still be moderate increases in RTT due to cumulative factors like increased distance, interference, and the complexity of multi-hop transmission.

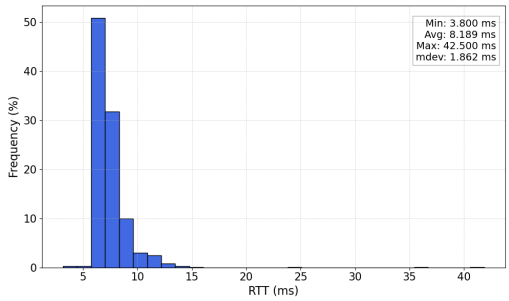
- **Double-Hop (One relay node)**

In this scenario, node 5 acts as a relay, enabling the connection between node 7 and node 10. The addition of this hop introduces delays due to the inherent processing and transmission required at the intermediate node. Specifically, the relay must process and forward the packet, which adds extra handling time. Furthermore, the packet may also be subject to retransmissions if any losses occur in the medium, further contributing to the overall delay.

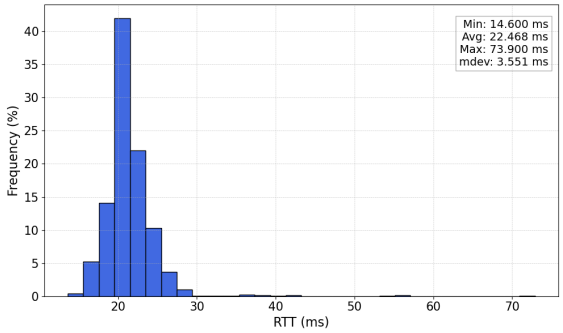
Even for small packets, a 2-hop network introduces higher latency and increased variability compared to P2P. The outliers in RTT may reflect congestion or link instability across the intermediate hop, though small packets still perform efficiently under normal conditions (figure 4.8a). The average increased by less than 2 ms.



(a) Ping of size 64 bytes



(b) Ping of size 1408 bytes



(c) Ping of size 5008 bytes

Figure 4.8 - RTT values of ping commands with different packet sizes in a 2-hop network

For medium-sized packets (figure 4.8b), the network handles these packets more consistently than small packets. Environmental interference led to an outlier in the case of the 64-byte packets, causing an unexpected spike in delay. Without the 73 ms outlier, both packet sizes of 64 and 1408 bytes experience some cases where the RTT can rise up to 50 ms. These spikes occur highly due to the relay node.

The results do not show a doubling of the P2P performance despite adding a relay, as one might expect. This is likely due to a significant ICMP processing overhead at both end points compared to the link transmission time.

The average RTT (22.468 ms, Figure 4.8c) is significantly higher than the results seen for 64-byte and 1408-byte packets. The large packet size requires fragmentation, which introduces additional overhead at each hop. The second hop further exacerbates the delay as both hops must process and reassemble the fragmented packets.

- **Triple-Hop (Two relay nodes)**

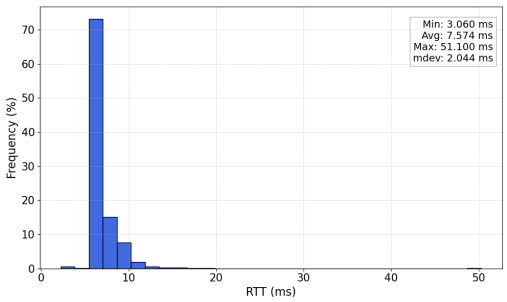
In this scenario, nodes 5 and 6 act as a relay to the communication between 7 and 10 (like the representation in Figure 4.4).

When comparing the three graphs (figure 4.9), a clear relationship emerges between the RTT distributions of the different packet sizes. The graph for 1408 bytes can be seen as a displacement of the 64-byte graph, with a slight increase in both latency and variance. Similarly, the graph

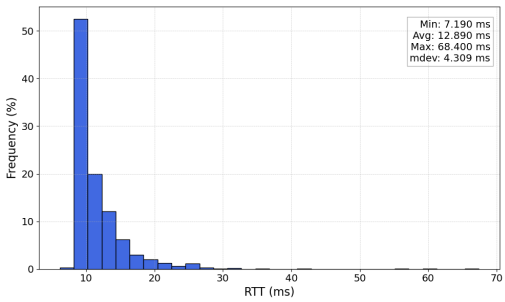
for 5008 bytes follows the same pattern relative to the 1408-byte graph but with an even more pronounced increase in both RTT and variability.

This suggests that as packet size increases, the overall distribution of RTT values shifts to the right, reflecting longer transmission times. However, the fundamental shape of the distributions remains relatively consistent. The primary difference lies in the extent of the shift and the magnitude of variance — larger packets not only take longer to traverse the network, but they also exhibit more inconsistent performance due to fragmentation and the additional processing overhead involved in handling them across multiple hops.

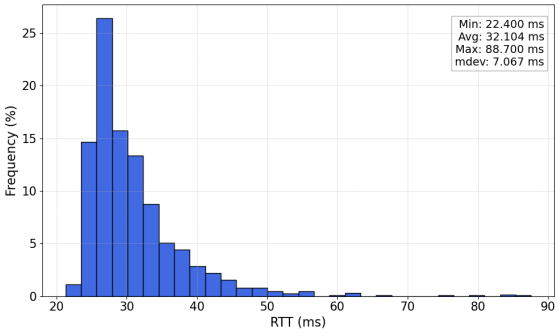
In essence, each successive increase in packet size results in a similar but more stretched distribution, where the key characteristics of the previous, smaller packet size graph are preserved but with greater delays and jitter.



(a) Ping of size 64 bytes



(b) Ping of size 1408 bytes

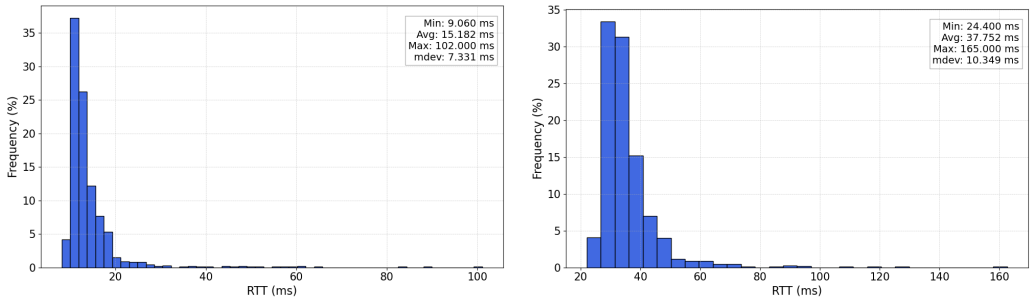


(c) Ping of size 5008 bytes

Figure 4.9 - RTT values of ping commands with different packet sizes in a 3-hop network

These findings confirm the trade-offs between packet size, hop count, and network performance. As the number of hops increases, the performance impact becomes more pronounced, especially for larger packets. These results serve as a critical reference point for understanding how multi-hop networks handle different packet sizes. These results highlight the cumulative impact of additional hops on RTT and variability.

The graphs with interference (figure 4.10) show a similar overall pattern to those without interference, with the key difference being the greater magnitude of values due to the delays introduced



(a) Ping of size 1408 bytes

(b) Ping of size 5008 bytes

Figure 4.10 - RTT values of ping commands with different packet sizes in a 3-hop network with interference

by the higher packet flow in the network. While the average and minimum RTT values remain relatively consistent in both scenarios, the main impact of interference is seen in the increased variance in delivery times.

Notably, the presence of interference introduces greater variability, with outliers becoming more frequent and, in some cases, exceeding 100 ms. This increased variance indicates that the network can struggle more under interference, likely due to retransmissions and packet collisions caused by the congestion in the medium. Despite the overall structure of the RTT distributions remaining similar, the interference amplifies the jitter and results in more pronounced fluctuations to higher values in network performance.

- **Dynamic Topology**

We conducted tests where the network topology was changed mid-testing. Figure 4.11 represents the RTT (Round Trip Time) values recorded during a ping test over a P2P network. As expected, the values are predominantly concentrated between 4-5 ms, which aligns with the anticipated performance for this configuration. The observed spikes in RTT could be attributed to interference in the environment as well as retransmissions initiated by the network interface to compensate for lost or corrupted packets. Figure 4.12 represents the RTT values for a 3-hop network. The average of the values goes up to 7-7.5 ms, and we can observe more inconsistency caused by the number of relays. Finally, Figure 4.13 represents a ping through the Multi-hop Network 1 configuration. The change observed around sequence number 40 corresponds to the route change. As the topology updates, the RTT values increase, reaching the levels shown in Figure 4.12. In this process, there were 3 packets lost.

The RTT has an average of 6.295 ms, with a minimum of 4.250 ms and a maximum of 14.400 ms, along with an average jitter of 1.525 ms. The network was able to maintain a stable connection, and the topology transition was not disruptive.

The Multi-hop Network 2 is shown in Figure 4.14. At sequence number 40, the network transitions to a 2-hop configuration, and around sequence number 90, it shifts to a 3-hop configuration. By examining the median RTT values, it is possible to identify which section corresponds to each

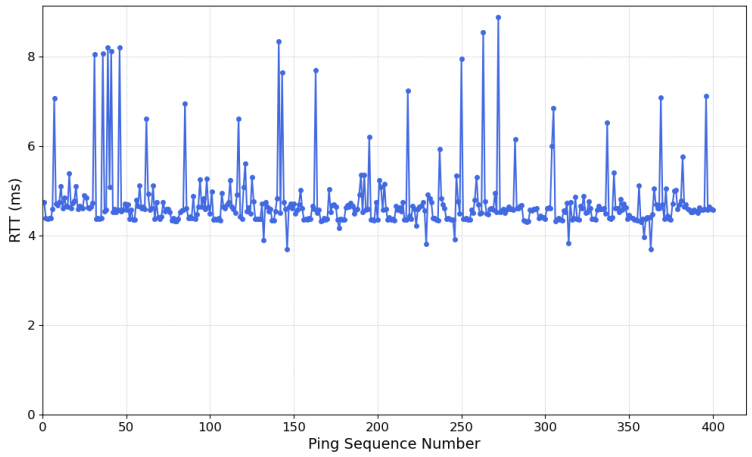


Figure 4.11 - RTT over a ping (64 bytes) sequence for a P2P topology

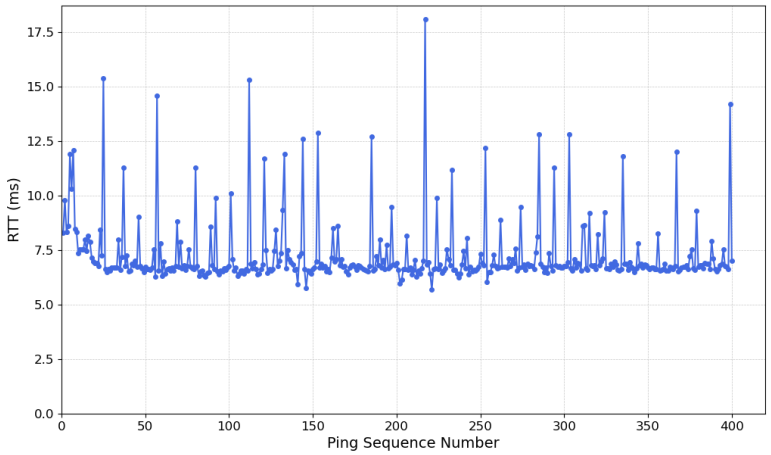


Figure 4.12 - RTT over a ping (64 bytes) sequence for a 3-hop topology

topology. Although the average RTT increases after each transition, the spikes remain contained within 12 ms.

In this test, a total of 5 packets were lost. For each topology change, it is possible to lose 1-3 packets. With the triple-hop configuration, the network exhibited an average RTT of 6.133 ms, a minimum RTT of 4.240 ms, a maximum RTT of 11.300 ms, and an average jitter of 1.244 ms.

While occasional packet loss is a trade-off, the flexibility to adapt the network topology ensures better overall performance and resilience in dynamic environments. This adaptability allows the network to efficiently handle varying conditions and maintain connectivity even when a direct P2P connection is not viable, ultimately enhancing the network robustness and responsiveness to changing demands.

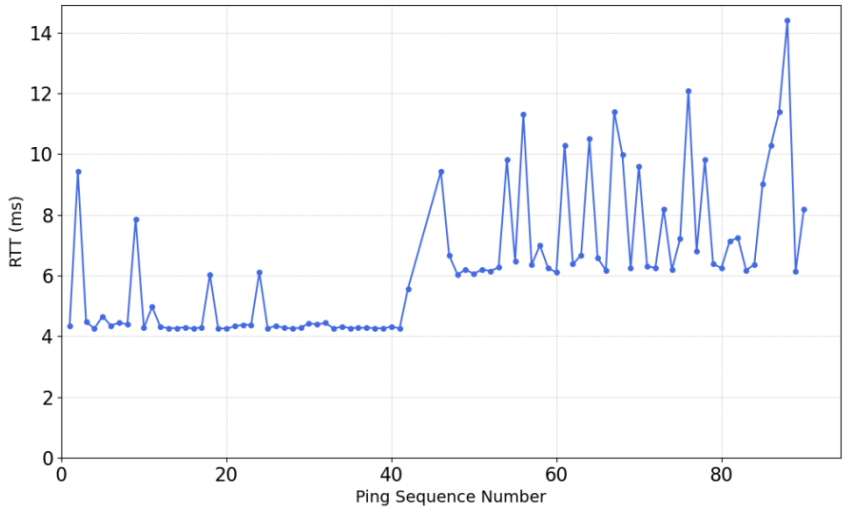


Figure 4.13 - RTT over a ping sequence a Multi-hop network 1

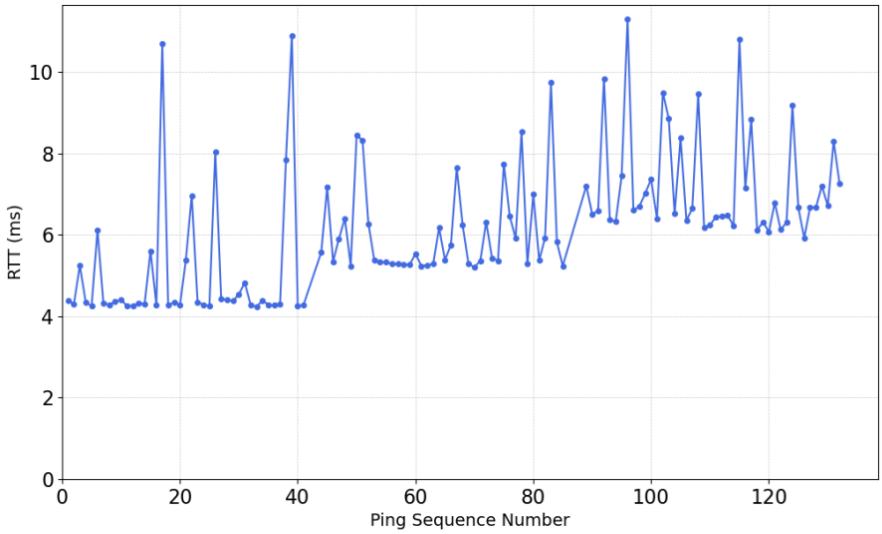


Figure 4.14 - RTT over a ping sequence a Multi-hop network 2

- **Conclusions**

Across all configurations, larger packet sizes (5000 bytes) consistently exhibit the highest jitter. This is largely due to fragmentation, which adds complexity and delay variation as multiple fragments are processed and reassembled at each hop.

As the number of hops increases, the jitter rises for all packet sizes. This suggests that multi-hop networks introduce more variability in packet delivery times.

High jitter is problematic for real-time applications like video streaming, voice calls, or gaming, where consistent packet delivery is crucial. The higher jitter values seen for large packets in 2-hop and 3-hop networks suggest that these configurations would struggle to support such applications without additional optimization or traffic management techniques.

Table 4.2 - Summary of metrics across different network configurations. These results are derived from tests with ping commands

	Bytes	Avg	Min	Max	Std	Packet Loss	Avg. Jitter
P2P	64	4.760	1.360	17.100	0.712	0.00%	0.480
	1408	5.438	1.770	28.500	1.169	0.00%	0.749
	5000	12.850	8.740	25.700	1.489	0.00%	1.310
2-Hop	64	6.455	2.210	73.400	2.929	0.00%	1.071
	1408	8.189	3.800	42.500	1.862	0.00%	1.105
	5000	22.468	14.600	73.900	3.551	0.00%	3.551
3-Hop	64	7.574	3.060	51.100	2.044	0.00%	1.332
	1408	12.890	7.190	68.400	4.309	0.00%	3.057
	5000	32.104	22.400	88.700	7.067	0.00%	6.162

A high jitter value is considered to be above 5 ms, which is only observed in the 3-hop topology with 5000-byte packets. This high jitter imposes a significant impact on network performance. It can cause noticeable issues in real-time applications, such as video and audio disruptions, excessive buffering, and delayed data transmission.

In all other configurations, the jitter value ranges from minimal (0-1 ms), which is typically ideal for most real-time applications. At this level, packet delivery times are highly consistent, and any variation is negligible, ensuring smooth performance. For other cases, jitter values fall within the medium range (1-5 ms), where the impact on network performance is moderate. Some real-time applications may experience slight degradation in quality, such as occasional stuttering in video or voice, but overall performance remains acceptable.

As explained in Section 4.3.4, the data packets from the video stream can contain up to 1400 bytes. Based on this analysis, we can conclude that video streaming in this multi-hop network is feasible. In scenarios with multiple hops, the stream can be maintained, though with slight performance degradation due to increased latency and variability.

4.4.2 TCP application Client-Server Results

In a multi-hop network, it is possible to establish and maintain a TCP connection. TCP plays a crucial role in stabilizing communication by ensuring packet order correction and retransmitting packets when necessary. In the TCP application, data is transmitted and received efficiently without causing saturation of the network medium. As a result, any packet losses are likely due to issues within the network medium itself, such as interference or weak signal strength, rather than network congestion. When

In both graphs (figures 4.15, 4.16), we observe the average throughput for a TCP Client-Server connection in a multi-hop network configuration. The throughput remains relatively steady until specific points where the network topology is updated, causing significant drops in throughput.

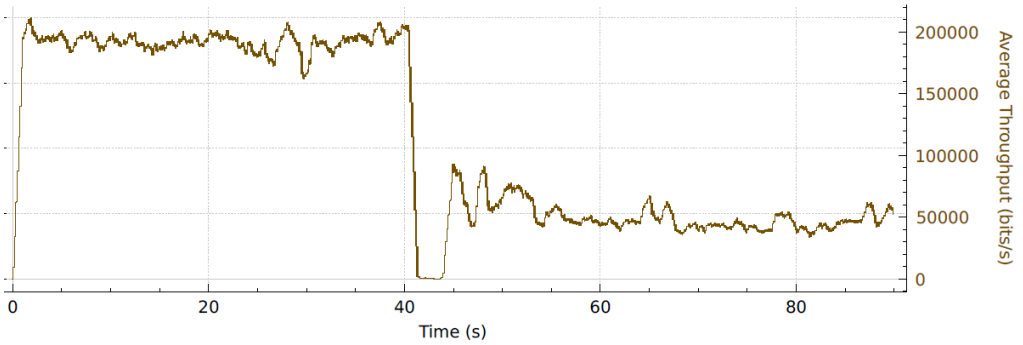


Figure 4.15 - Average throughput in a TCP Client-Server connection for Multi-Hop Network 1

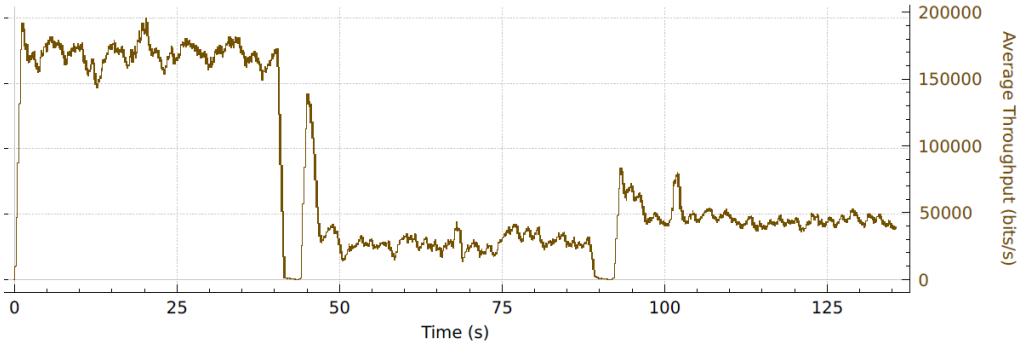


Figure 4.16 - Average throughput in a TCP Client-Server connection for Multi-Hop Network 2

When the network is updated, the throughput temporarily drops to zero, indicating that the connection is momentarily interrupted. This behavior occurs because the route change in the multi-hop network briefly cuts off the communication between the client and server. During this period, the TCP connection experiences a disruption, as the protocol requires time to adjust to the new topology and re-establish the connection.

Once the network update is completed and the connection is restored, the throughput gradually recovers, but not necessarily to its original levels. The drop in throughput after the update is due to adding additional hops in the new topology, which reduces the overall transmission efficiency.

The patterns shown in these graphs highlight the sensitivity of TCP throughput to network topology changes. Although TCP is designed to handle such disruptions by resending lost packets and adjusting the transmission window, the recovery process takes time, which is evident from the periods of low or unsteady throughput following the topology updates.

4.4.3 TCP connection via iperf3

In *iperf*3, throughput is determined based on the behavior of the TCP congestion control mechanism. TCP adapts its transmission rate according to network conditions to find the optimal throughput without overwhelming the network. This command dynamically tries to push as much data as possible through the network, up to the limit of the network capacity or the protocol limits. The goal is to saturate the link to observe the maximum performance under the given network

conditions. The results presented reflect both the network inherent bandwidth and its response to congestion and packet loss, with TCP adjusting to maintain a stable connection

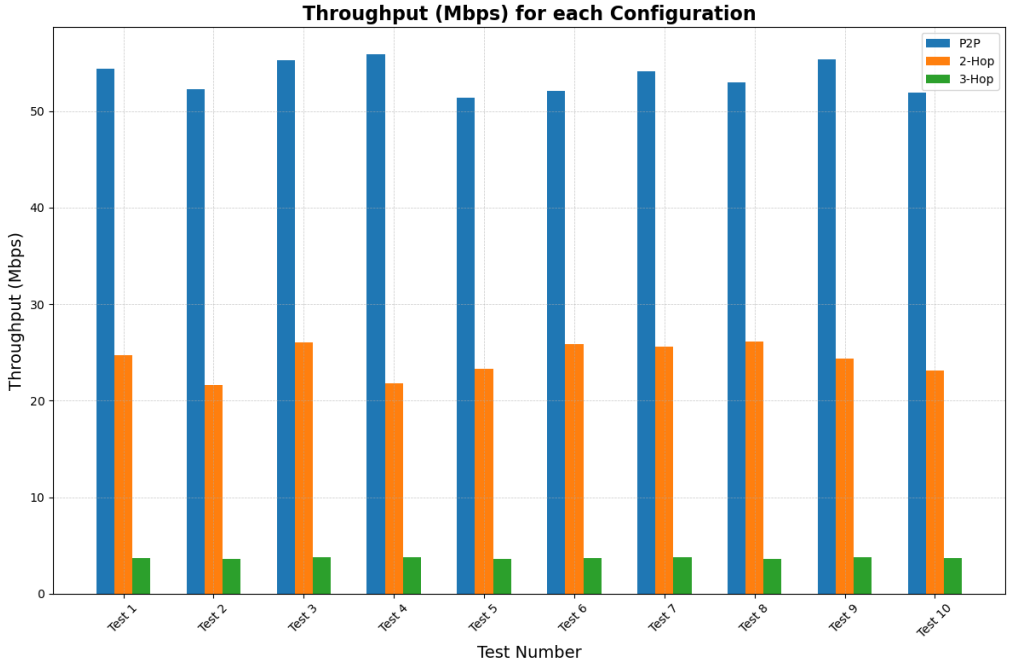


Figure 4.17 - Maximum values of throughput obtained with *iperf*3, for each configuration and test

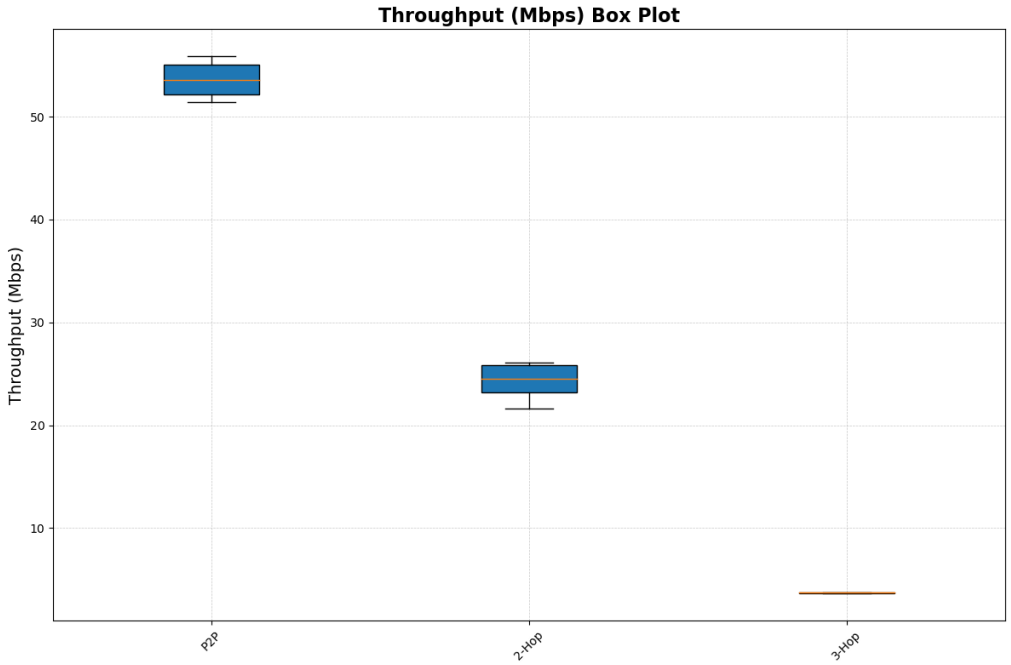


Figure 4.18 - Average values of throughput obtained with *iperf*3 for each configuration

The results in 4.17 are consistent across all ten test iterations, indicating that the behavior of the network remains stable under different conditions. The throughput values for each configuration

(P2P, 2-hop, and 3-hop) do not vary significantly from test to test, suggesting that the network performance is relatively predictable based on its topology.

As more relay nodes are introduced, the network performance degrades further. Each additional hop introduces more latency and packet processing delays and increases the likelihood of packet retransmissions, all of which significantly reduce the available bandwidth for data transmission. With the introduction of one relay, the throughput drops to nearly half of its original value. Inserting the second relay causes the throughput to degrade significantly more, to about 1/5 of the one relay value (figure 4.18).

No synchronization protocol was employed in these experiments; only CSMA/CA was used to manage communication interference and collisions. Due to limitations on the testbed, the nodes share the same collision domain. In the absence of synchronization, nodes may transmit data simultaneously, resulting in collisions. This leads to multiple nodes competing for access to the same communication medium. When packets collide, they must be retransmitted, consuming bandwidth and ultimately reducing overall throughput.

In an ideal testbed, there would be nodes that could not communicate with one another, significantly reducing the occurrence of collisions. For example, in a linear topology such as 1-2-3, transmissions from node 1 would not collide with simultaneous transmissions from node 3, allowing for more efficient communication.

In summary, the lack of synchronization in the network results in higher collision rates, increased contention for access, and greater latency. These factors collectively contribute to a significant reduction in throughput as the number of communicating nodes increases in a multi-hop network.

The 2-hop configuration in 4.18 shows moderate throughput (around 20-25 Mbps). While this throughput is lower than P2P, it is still adequate for medium to high-quality video streams.

The defined video stream settings — 320x240 resolution, 15 fps framerate, and a bitrate of 300 kbps (0.3 Mbps) — constitute a low-quality video stream. This configuration requires significantly less bandwidth than higher-quality streams such as HD or 4K video. With a bitrate of 0.3 Mbps, the video stream operates well within the bandwidth capabilities of all network topologies tested, including the 3-hop configuration. The combination of low resolution, framerate, and bitrate ensures stable performance across the network, even in the 3-hop setup.

4.4.4 Video Stream Results

To accurately evaluate the performance of the multi-hop stream in Wireshark, it is crucial to account for the forwarding of the same packet through the network. As packets are forwarded across multiple hops, they may appear repeatedly in the capture. To focus on actual data delivery and eliminate the effects of retransmissions, a filter was applied to capture only the packets successfully delivered to the destination node. By filtering based on the destination MAC address, the analysis provides an accurate reflection of the network's effective throughput, latency, and jitter without interference from retransmissions, intermediate forwarding, or repeated packets.

Table 4.3 - Summary of video stream metrics for different configurations: TCP vs. UDP

	Configuration	Decoded Blocks	Displayed Frames	Lost frames	Demuxed data size (KiB)	RTP Packets	Loss (%)
TCP	3-Hop	2840	1754	282	4059	3901	0.00%
	Multi-Hop Net. 1	2231	1428	218	3245	3167	1.01%
	Multi-Hop Net. 2	3102	2118	353	4466	4440	2.97%
UDP	3-Hop	2981	1822	305	4255	4090	0.07%
	Multi-Hop Net. 1	2228	1333	193	3256	3147	0.67%
	Multi-Hop Net. 2	3079	1925	305	4407	4345	2.90%

In a multi-hop network, the choice of transport protocol plays a critical role in shaping the performance and perceived quality of video streaming. When using UDP, the video stream tends to flow smoothly under typical conditions, but it may occasionally experience minor glitches. These issues can manifest as blurry sections, pixelation, or dropped frames, particularly in moments of network instability. However, these visual artifacts are usually brief and do not interrupt the overall continuity of the stream. UDP, being a connectionless protocol, does not guarantee packet delivery or order, but it excels in low-latency applications. Although the visual quality may degrade slightly under high loss rates, the protocol ability to push packets through the network without waiting for retransmission ensures that the stream remains unbroken. In practice, this results in a video stream that prioritizes continuous delivery over perfect fidelity.

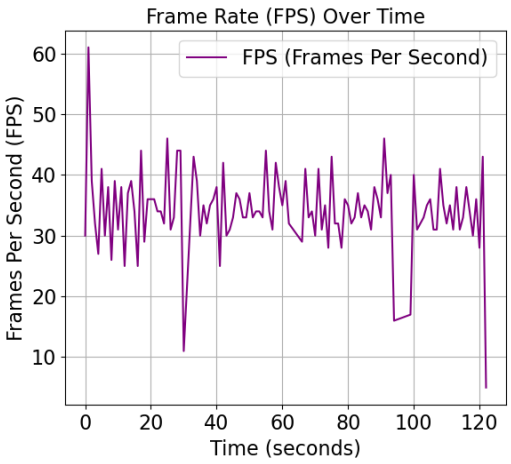
In contrast, TCP offers a more reliable but potentially disrupted video streaming experience. TCP ensures that all packets are delivered and arranged in the correct order, which means that under stable network conditions, the video stream exhibits high quality with no visible loss of frames or degradation. However, this reliability comes at a cost, particularly in multi-hop networks where jitter, delays, and packet loss can compound.

When packet loss occurs in a TCP stream, the protocol initiates retransmissions to recover the missing packets. This process introduces buffering and pauses in the video, as the receiver must wait for the lost data to be resent. As a result, while the visual quality remains intact, the viewing experience is often disrupted by longer pauses and higher latency. In multi-hop networks, where packet loss and delays are more likely due to the increased number of hops, TCP's retransmission mechanisms can cause frequent freezing of the video stream, especially during periods of network congestion or instability.

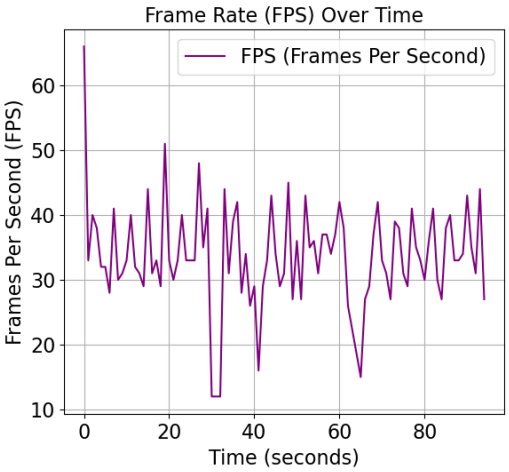
In the Multi-Hop Network 1, the topology changes at 40 seconds. For Multi-Hop Network 1, changes at 40 and 80 seconds.

In the TCP frame rate, 4.19 is more variant as we add more hops. The 4.19b shows a less variable framerate up until the 40-second mark, at which point it was in a P2P topology. In the 4.19c, between 40-80 seconds, it is re-establishing the connection. After the 80-second mark, the framerate has a lot more variance. The breaks observed in the video stream are directly correlated to moments when the stream freezes due to buffering in VLC. These breaks occur as the VLC player loads additional data into its buffer, temporarily halting playback until sufficient data is available to resume the video stream smoothly.

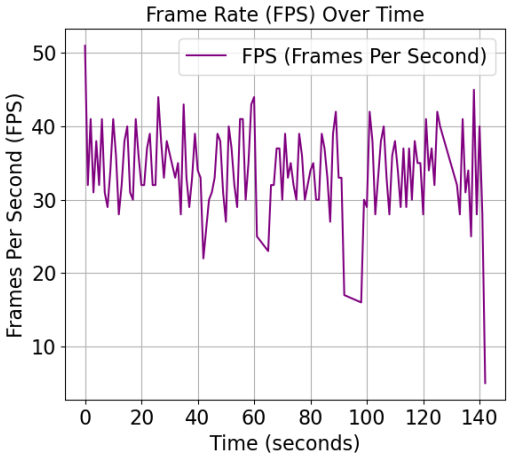
For the UDP, 4.20, we can see drops to almost 0 frames. Some of these drops are also related to the changes applied in the routes. This highlights a key distinction between TCP and UDP



(a) 3-hop network



(b) Multi-Hop Network 1



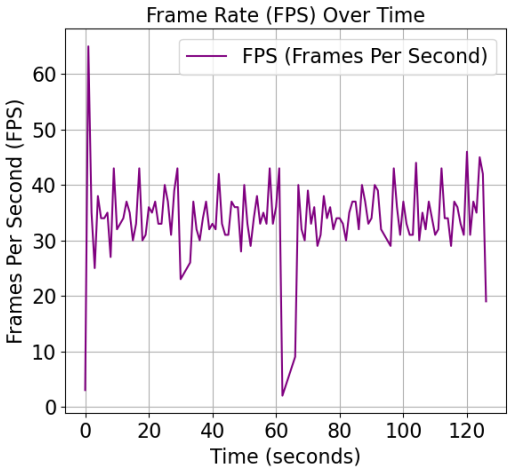
(c) Multi-Hop Network 2

Figure 4.19 - Frame Rate over time, for different topologies, using TCP protocol

in handling network interruptions. When there are packet losses or interference, UDP struggles to maintain a stable connection because it does not implement error correction or retransmission mechanisms.

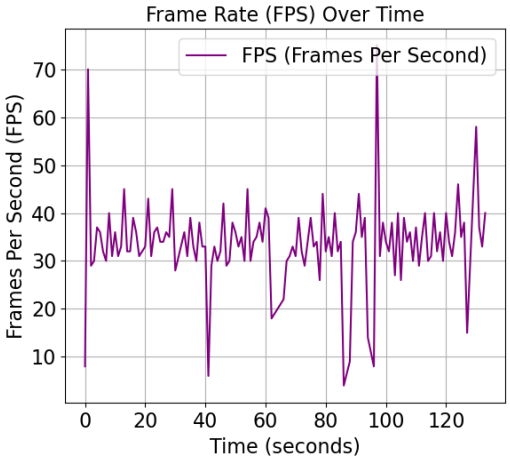
The decided block refers to the number of data blocks successfully decoded by the video player. Each block contains part of the video data necessary for rendering the stream. The demuxed data sie corresponds to the amount of video data extracted from the stream for processing and playback.

Observing the Table 4.3, as expected, UDP exhibits some packet loss in all scenarios, as it lacks the built-in retransmission mechanisms that TCP offers. However, TCP encounters higher packet loss, particularly during dynamic topologies. This is likely due to momentary connection losses when routes are being exchanged or updated. The constant changes in topology introduce interruptions that negatively impact TCP's performance, as it relies on stable connections to ensure



(a) 3-hop network

(b) Multi-Hop Network 1



(c) Multi-Hop Network 2

Figure 4.20 - Frame Rate over time, for different topologies, using UDP protocol

reliable data delivery. This contrast highlights the trade-off between the two protocols: while UDP sacrifices reliability for speed and low overhead, TCP suffers more in environments with frequent topology changes due to its need for sustained connection stability.

Both TCP and UDP experience degradation in performance as the network topology becomes more complex. In multi-hop networks, packet loss increases, which in turn affects the number of displayed frames and decoded blocks. TCP is more affected by this degradation due to its reliance on retransmissions, while UDP maintains a more stable stream at the cost of some quality loss (as evidenced by the higher lost frames).

4.5 Results With TDMA

In this section, we analyze the results obtained when using the TDMA framework with the RA-TDMAs+ synchronization.

4.5.1 Ping Command Results

The RTT represented by pings does not reflect the actual transmission time of the packet, as there is additional processing overhead that causes delays. When an ICMP Echo Request packet is generated, a timestamp is embedded in the payload of the ICMP message. This timestamp records the exact time the request is sent by the source device and is usually added at the network stack level (IP/ICMP layer) within the operating system just before the packet is sent to the NIC.

In the TDMA structure, this packet is first routed to the tun0 interface before reaching the NIC. There, it undergoes the TDMA header encapsulation process and waits in the buffer for the appropriate TDMA time slot. At the receiver, the packet follows the reverse process: it is extracted from the buffer, the TDMA header is removed, and the packet is then delivered to the network stack level (IP/ICMP layer) to generate the ICMP Echo Reply. This reply follows the same process on the return path.

This underlying process, which is not visible to the ICMP protocol, can increase the RTT values recorded by ping. However, these increased RTTs do not necessarily indicate degraded TDMA performance, nor do they suggest that the time slots and rounds in the TDMA system are not being respected.

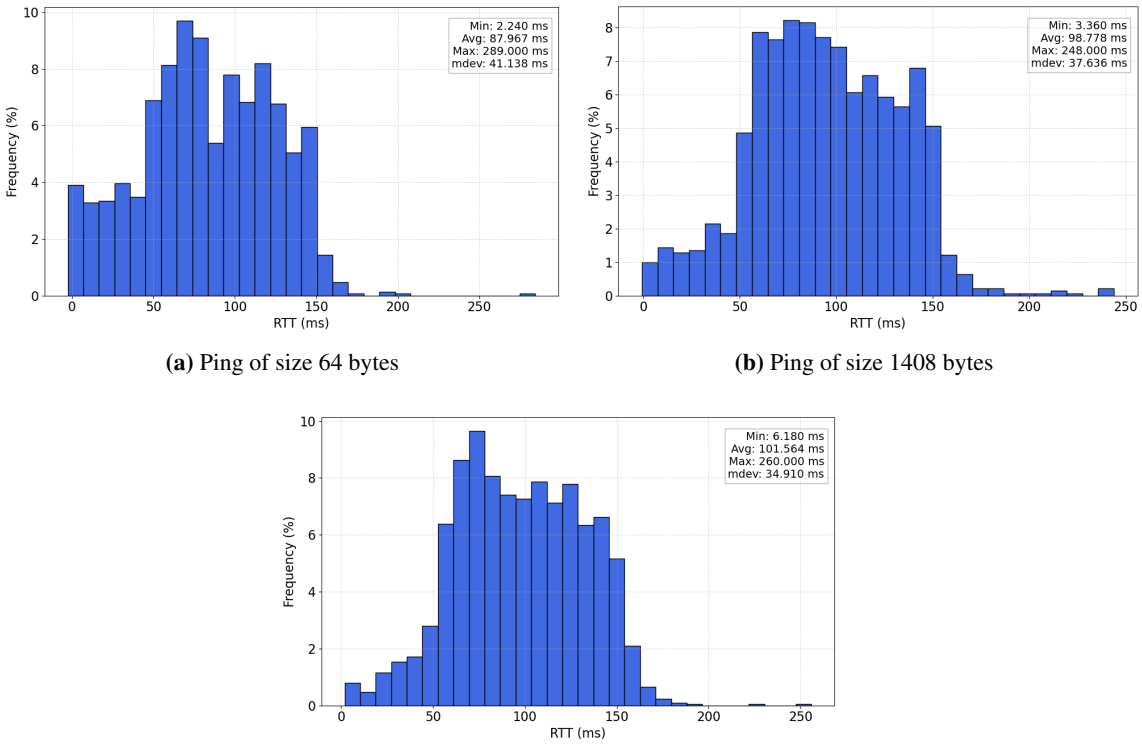
These RTT values reflect the network-layer perspective of the ICMP protocol.

All RTT graphs show a high variance in the results. This is due to the fact that packets are sent within the duration of the slot, causing RTT values to vary significantly based on how long a packet must wait before transmission. Ping packets generated outside the time slot experience longer waiting times than those generated and sent just before the slot ends. These latter packets have the shortest RTT values, as they are transmitted immediately at the start of the next available slot.

- **P2P (Direct communication)**

All of the graphics in 4.21 show some outliers in the order of 250 ms. These results could represent instances where the group was synchronized, but the interface performed multiple retransmissions due to interference and the resulting delays. If a packet transmission is delayed, it will also delay the transmission of the synchronization packet, potentially disrupting the overall synchronization state. The low RTT values likely represent cases where the packet was sent almost immediately by the sender (node 10), having spent little time in the buffer, and reached the destination slot (node 7) right after. In these instances, node 7 transmitted the packet without significant delay.

The average values indicate that even accounting for processing and buffer waiting times, a ping is sent and received in less than one round.



(a) Ping of size 64 bytes

(b) Ping of size 1408 bytes

(c) Ping of size 5008 bytes

Figure 4.21 - RTT values of ping commands with different packet sizes in a P2P network, with TDMA

With interference (figure 4.22), the graph 4.22a has a packet loss of 0.00% and the graph 4.22b of 0.42%. The RA-TDMAs+ synchronization can handle the interference as the packet loss is near zero. The minimum and average values are similar to those obtained in tests without interference. The key difference, however, is that the RTT values are now more distributed between the minimum and the maximum.

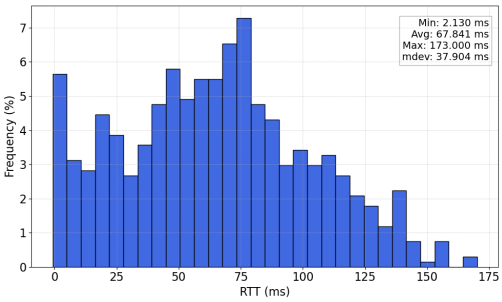
- **Double-Hop (One relay node)**

This topology exhibited higher packet loss compared to the other topologies (Table 4.4).

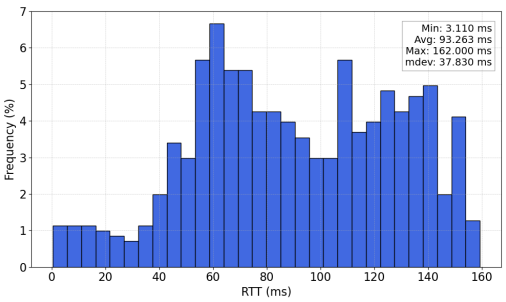
In this topology, we can also conclude that the average RTT, from the moment the source sends the request to when it receives the reply at its interface, is practically less than one round. The standard deviation follows a similar pattern to the previous case, with the 5008-byte packets exhibiting the highest value and the most significant variation (figure 4.23).

- **Triple-Hop (Two relay nodes)**

With a triple-hop configuration and synchronization, it was possible to achieve minimal RTT values that were approximately equivalent to the minimum values observed in the same testbed without TDMA(figure 4.24).

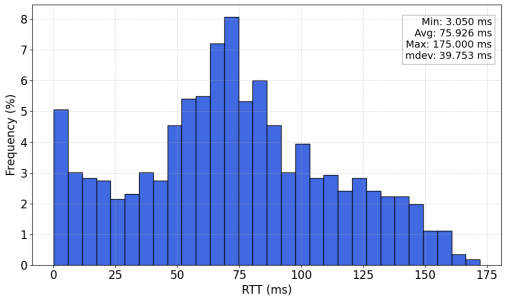


(a) Ping of size 64 bytes

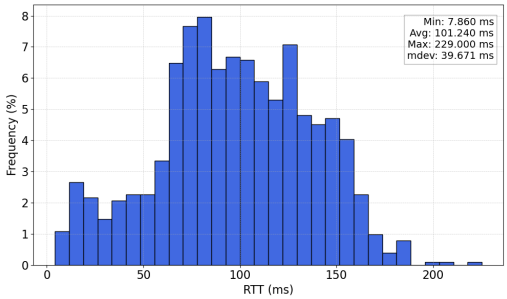


(b) Ping of size 1408 bytes

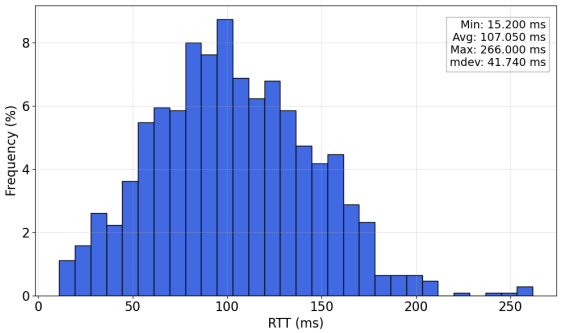
Figure 4.22 - RTT values of ping commands with different packet sizes in a P2P network with interference



(a) Ping of size 64 bytes



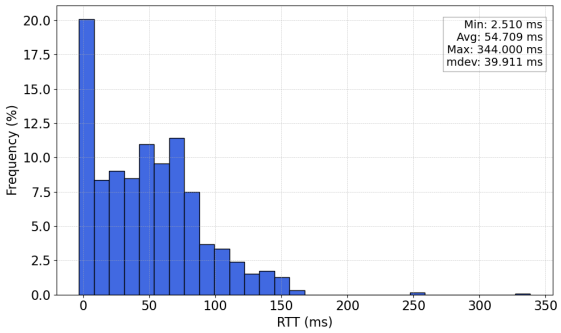
(b) Ping of size 1408 bytes



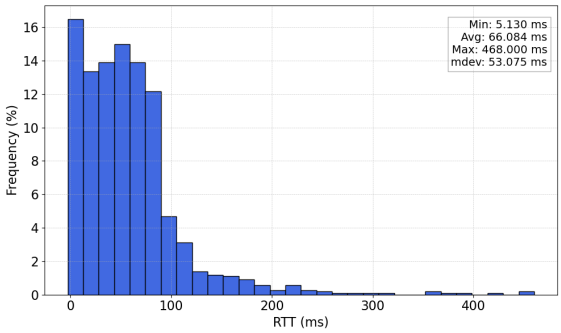
(c) Ping of size 5008 bytes

Figure 4.23 - RTT values of ping commands with different packet sizes in a 2-hop network, with TDMA

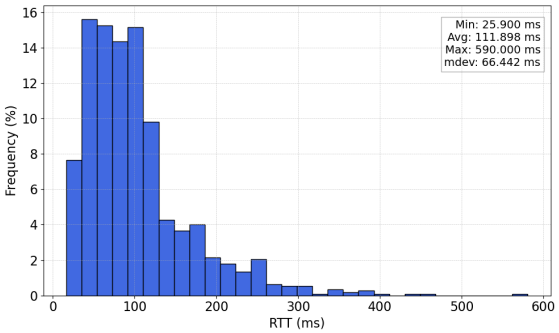
The values from 0-100ms are the most prominent in all three cases. However, this is the topology that shows the highest RTT for outliers, being from 300-600ms, which represents 3 to 6 TDMA rounds.



(a) Ping of size 64 bytes



(b) Ping of size 1408 bytes



(c) Ping of size 5008 bytes

Figure 4.24 - RTT values of ping commands with different packet sizes in a 3-hop network, with TDMA

For the interference tests (figure 4.27), for each test case, there is a total packet loss of 0.00%, 11.29%, and 12.82%. As the packet size increases, noticeable gaps appear between the RTT values, a pattern that was not observed in the other cases. This combined with the packet loss, indicates that it becomes more challenging to maintain organized communication when there is a large volume of packets circulating within the network.

We used Wireshark to measure the time a packet takes to be sent and received through the *wlan*0 interface. These values exclude the processing and buffer waiting times in the TDMA framework at the sender. However, they include the processing time at the relay nodes, as well as the time spent in the TDMA framework and network layer at the receiver to generate and send the reply. Once the destination node receives the Echo Request, it processes the packet, generates the ICMP Echo Reply, and sends it back to the tun0 interface, which will go through the TDMA framework process.

The Figure 4.25 confirms that in a 3-hop network, by excluding the time spent waiting in the TDMA buffer, it is possible to exchange pings without exceeding a full TDMA round. In contrast, Figure 4.26 represents the same test but includes the time taken for the packet to return to the

source's wlan0 interface from the destination. The difference between these two graphs is add up to 20 ms. Notably, only two instances in the test exhibit an RTT exceeding the round period.

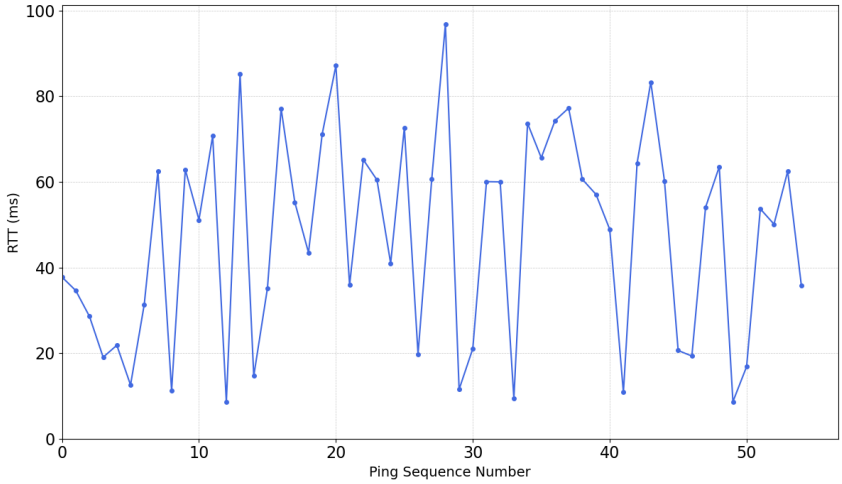


Figure 4.25 - In a 3-hop topology, these RTT values represent the time taken from when the request packet enters the *wlan*0 interface until the destination sends the reply. This includes processing times in the relays and in the destination node.

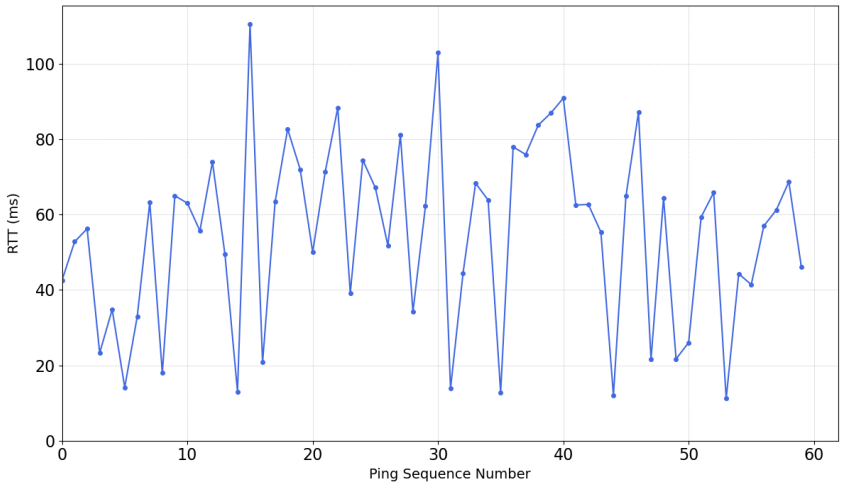
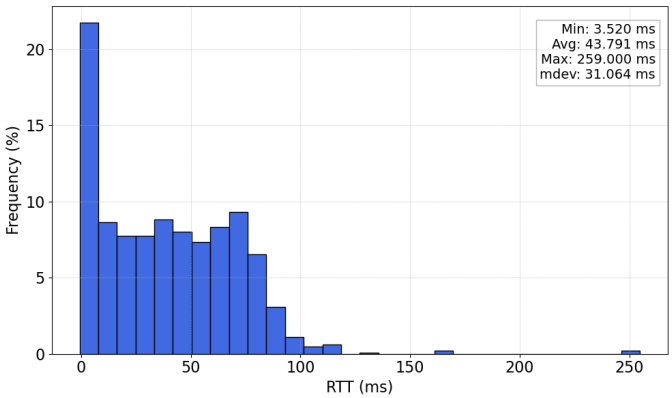


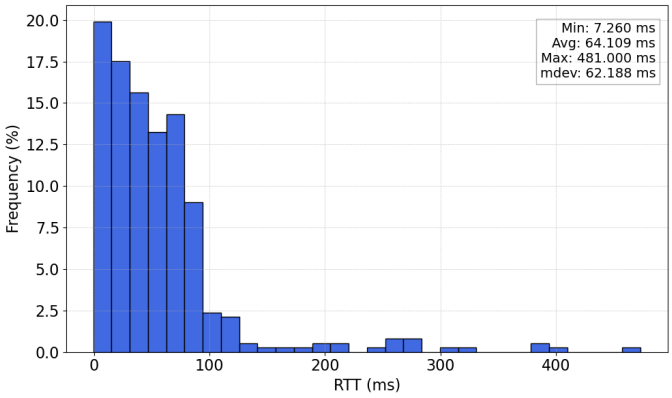
Figure 4.26 - In a 3-hop topology, these RTT values represent the time taken from when the request packet enters the *wlan*0 interface; the destination sends the reply up to the reception in the *wlan*0. This includes processing times in the relays and in the destination node.

- **Dynamic Topology**

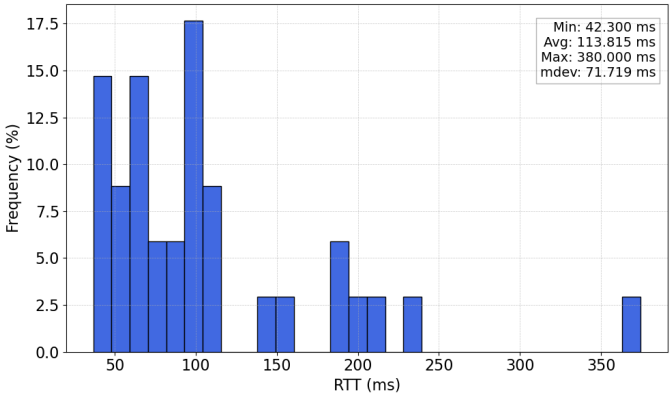
The graph 4.28 illustrates an interesting behavior in the network. Up to sequence number 40, the configuration is P2P, with an average RTT of approximately 75 ms. Just before the topology update, a slight inconsistency can be observed, likely due to synchronization processes within the network.



(a) Ping of size 64 bytes



(b) Ping of size 1408 bytes



(c) Ping of size 5008 bytes

Figure 4.27 - RTT values of ping commands with different packet sizes in a 3-hop network with interference

After sequence 40, the configuration switches to 2-hop, where RTT values increase, reaching a peak before dropping abruptly—a pattern that repeats multiple times. Starting at sequence 80, a similar behavior is noted, but with more abrupt fluctuations, resulting in significantly higher RTT values.

During this test, the network exhibited an average RTT of 68.029 ms, with a minimum of

2.860 ms, a maximum of 175 ms, and an average jitter of 26.769 ms. No packet loss was recorded during the test.

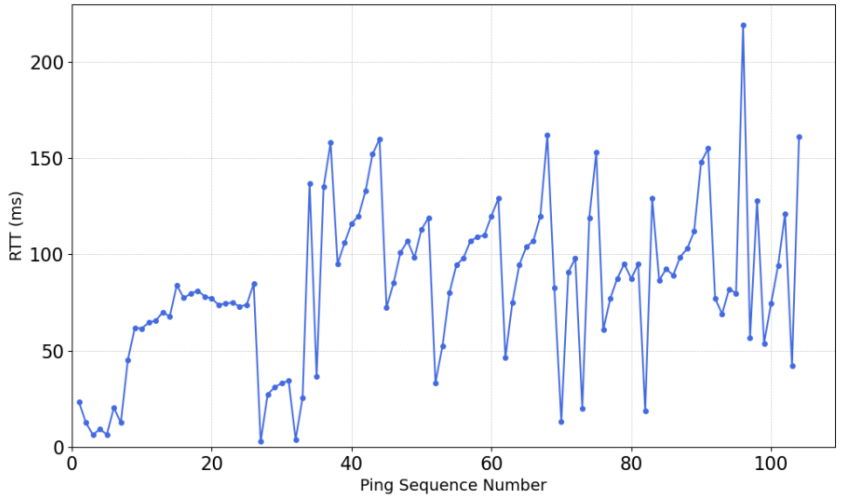


Figure 4.28 - RTT values of a ping (64 bytes) traveling the Multi-Hop Network 2, with TDMA framework running

The compartment mentioned before can be observed in both figures: for P2P 4.29, and 3-hop 4.30. The graph for the 3-hop network has a greater divergence between the maximum and minimum RTT values, making it more challenging to identify a clear periodic behavior.

In the P2P graph, the maximum RTT values occur when packets are sent at the beginning of the TDMA round. In this situation, the packet must wait longer for the response, as it has to pass through the entire round of TDMA slots before reaching the destination and returning. As the round progresses, the cumulative processing and sending delays cause the pings to be sent later in the time slot, gradually reducing the wait time for the response.

The later a packet is sent within its designated slot, the quicker the response will be received, resulting in lower RTT values. This cyclical behavior creates fluctuations in the RTT values. These fluctuations represent the variability in how long packets wait in the TDMA buffer before being sent. Spikes in RTT occur when packets are delayed waiting for their next available slot, while lower RTT values happen when packets are sent immediately within their allocated slot.

This pattern is a direct consequence of the TDMA structure, where each node must wait for its assigned slot to transmit. The observed variability in RTT reflects how efficiently packets are processed depending on their timing within the TDMA round.

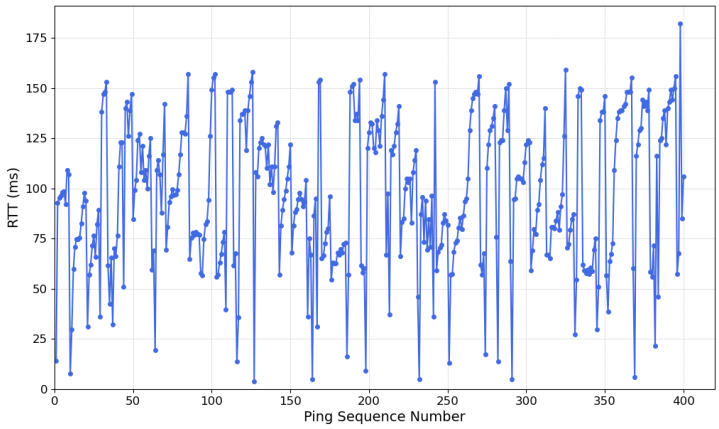


Figure 4.29 - RTT values of a ping (64 bytes) traveling a P2P network, with TDMA framework running

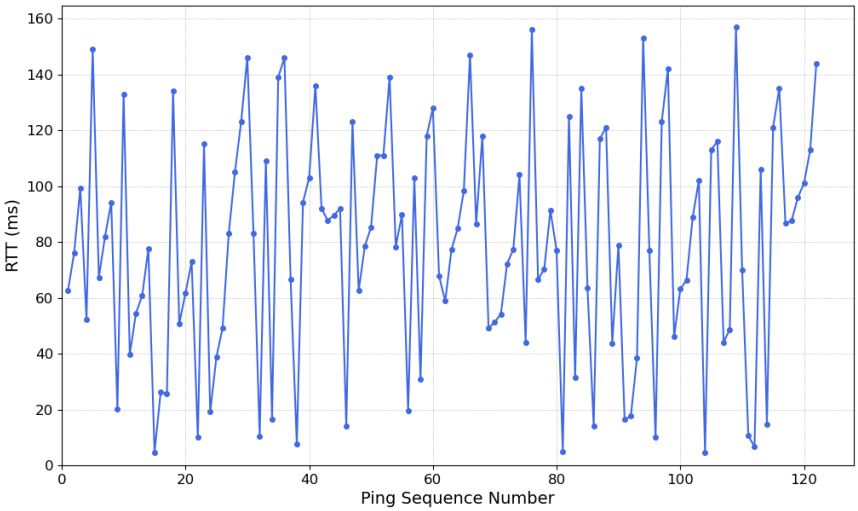


Figure 4.30 - RTT values of a ping (64 bytes) traveling a 3-Hop network, with TDMA framework running

- **Conclusions**

All packets traveling through the *wlan*0 interface are UDP packets, as they are transmitted via the UDP socket of the TDMA framework. Using Wireshark, we can identify which packets correspond to an ICMP request and reply by analyzing their IP addresses and MAC addresses. By using Wireshark's tools and a Python script, we can determine the time it took for the respective ping to be sent from the source node *wlan*0 interface, the time when the destination node replies, and when those packets arrive back at the source node *wlan*0 interface. It should be noted that these calculated times include the processing time of the ICMP packet by the TDMA framework and the network layer, but only at the destination.

The topologies may change, but the transmission times and their order remain the same, meaning that how the round is organized and how the route is laid out can influence overall performance. In any topology, as soon as node 7 finishes its slot, node 10 can begin sending its replies, allowing for relatively low RTT values that are close to the average observed in experiments without TDMA. However, if node 10 does not complete sending all of its replies within its slot, it must wait approximately 75 ms before continuing.

In a 2-hop topology, with node 5 acting as the relay, node 5's slot starts immediately after node 10. If there are any delays in packet transmission at the kernel level, this can cause congestion and place stress on the network, resulting in additional delays and influencing the synchronization protocol. In a 3-hop topology, the transmission load is more evenly distributed across the two relay nodes, which helps to better manage the transmission load and minimize congestion.

A triangular shape becomes more prominent in the histograms of RTT values, being more prominent in the P2P and 2-hop topology. A triangular shape in an RTT histogram indicates consistent variability with uniform distribution across a range of RTT values, reflecting a balance between low and high RTTs without extreme clustering. This could result from jitter, buffer waiting times, or network delays spread relatively evenly over time.

On average, it is possible to exchange packets with a node and receive a response in slightly less than one round in any configuration. We can also conclude that the TDMA framework can perform similarly even under interference for P2P and 2-hop networks, effectively adapting to the high flow of packets between nodes. In the case of 3-hops, the entire network becomes congested, which makes communication and synchronization more difficult.

The packet loss observed (Table 4.4) in the 2-hop topology is likely due to the increased interference caused by the P2P communication relying on a single relay node. This relay has its transmission slot positioned between the source and destination, which can lead to more delays and interference. Such interference can be influenced by environmental factors, including retransmission attempts by the wireless system, which in turn can cause delays in the transmissions necessary for synchronization.

The average jitter (Table 4.4) is approximately 20-30 ms, indicating that the median time difference between consecutive packets is about one round. This suggests that, on average, packets can be delivered within the sender's slot, even with relays along the route. However, in the case of 5008-byte packets for a 2-hop network and 1408-byte packets for a 3-hop network, the jitter nearly approaches two rounds, which may cause some interference during the relay process. The worst case is with 5008-byte packets in a 3-hop network, where the high number of retransmissions results in the greatest delay and interference, with jitter reaching almost three rounds. In this case, a single ping request generates the transmission of a total of 12 packets. The source node initiates communication by sending a ping request, which is fragmented into four packets. At each relay node, these packets are reassembled and subsequently routed to the next destination according to the routing table. Upon reaching the final destination, the network has processed a total of 12 packets.

Table 4.4 - Summary of metrics across different network configurations with TDMA framework, using the ping command

	Bytes	Avg	Min	Max	Std	Packet Loss	Avg. Jitter
P2P	64	87.967	2.240	289.000	41.138	0.00%	31.939
	1408	98.778	3.360	248.00	37.636	0.00%	31.921
	5000	101.564	6.180	260.000	34.910	5.92%	23.575
2-Hop	64	75.926	3.050	175.000	39.753	0.94%	32.832
	1408	101.240	7.860	229.000	39.671	14.88%	37.863
	5000	107.050	15.200	266.000	41.740	11.65%	41.084
3-Hop	64	54.709	2.510	344.00	39.911	1.65%	32.322
	1408	66.084	5.130	468.000	53.075	1.36%	44.590
	5000	111.898	25.900	590.000	66.442	1.32%	63.061

4.5.2 TCP connection via iperf3

In the TDMA framework, we focused on analyzing throughput performance. In the P2P configuration, Fig. 4.31, there are some outliers that reach up to 16 Mbps, but the average throughput typically falls between 4-8 Mbps. When the network operates in a 3-hop configuration, throughput decreases to around 2 Mbps.

Each node is permitted to transmit exclusively within its designated time slot. As a result, each node effectively has access to only one-quarter of the maximum available bandwidth, explaining the lower values. This is in contrast to the tests conducted without this structure, where each node can utilize the full bandwidth for establishing communications. Additionally, it is important to note that the time slots operate conservatively, allowing each node to utilize approximately 50% of the available communication time within its slot. This approach is necessary to ensure that all packets transmitted within the slot have sufficient time to reach their destination before the slot ends. However, this does not take into account the potential delays associated with multi-hop communications.

The TDMA framework imposes a strict queue time organization, handling a large number of packets sent simultaneously. Despite the lower throughput in multi-hop scenarios, this performance is still sufficient for transmitting video streams. The synchronization inherent to the TDMA framework helps manage interference, making it more efficient for controlled, low-bitrate video transmission.

4.5.3 Video Stream Results

Even with delays, jitter, and low throughput, the network is still capable of maintaining a video stream. The TDMA framework, despite introducing some latency, ensures controlled transmission timing, which helps to mitigate interference and maintain consistent communication. The low-bitrate requirements of the video stream fit within the available throughput, allowing for stable streaming even in multi-hop configurations with lower performance.

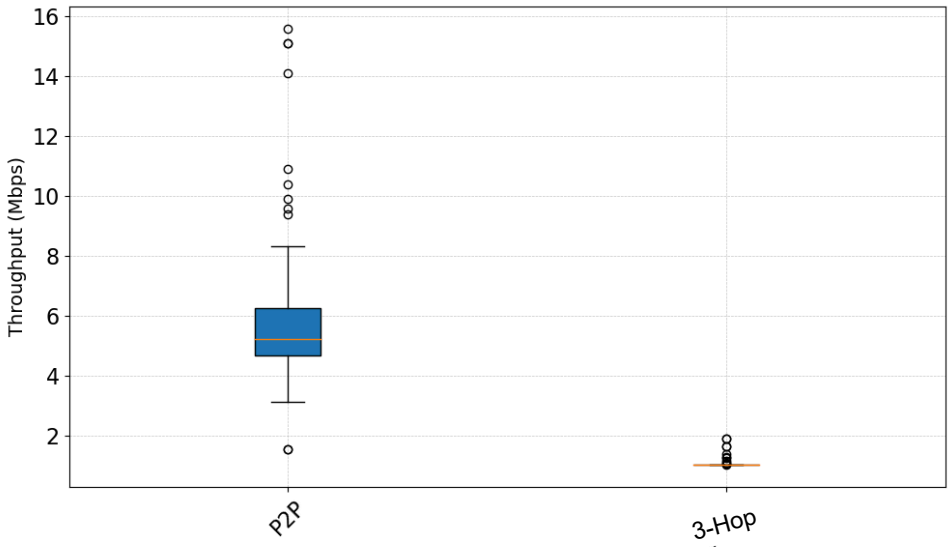


Figure 4.31 - Average values of hroughput for the P2P and 3-hop topologies, obtained with *iperf*3.

All stream packets are encapsulated through the TDMA framework. Thus, we only see UDP packets traveling the air. We tested the TDMA framework with a stream using a TCP connection. Since the TCP connection is encapsulated within UDP, it can help provide a more stable connection by enabling retransmissions when necessary, mitigating potential packet loss.

In TCP, packet loss typically triggers retransmissions to ensure that all data is eventually received. However, in real-time streaming applications, the delay introduced by retransmissions can lead to degraded video quality, causing frames to appear corrupted or out of order, which explains the distorted image. This phenomenon is presented in Figure 4.33.

This issue is sometimes also described as "compression artifacts" or "glitching" when it results from how video codecs handle missing data. In this case, it likely relates more to packet loss and retransmission delays in the network connection.

When the video stream begins, the player (e.g., VLC) first buffers a portion of the video before starting playback. Buffering involves downloading and storing some video data in advance, allowing the player to compensate for network variability, such as jitter or packet loss. Since TCP requires all packets to arrive in order, any delayed or lost packet prevents the buffer from filling properly until the missing packet is recovered. During this time, the video will freeze as the buffer waits for more data to resume playback.

In a video stream, even if a single packet is delayed or lost, TCP stops sending video data to the application until the missing packet is recovered. This causes buffering delays, where the video pauses while waiting for the missing data to arrive. As the buffer fills up, VLC collects more data to compensate for potential future delays. However, the buffering itself introduces lag, as TCP waits for all packets, including retransmissions, to arrive before continuing the stream.

The video stream from this test can be seen in the following URL: `https://www.youtube.com/watch?v=UIqxExueg9w`. We can see the effects previously described in the video.

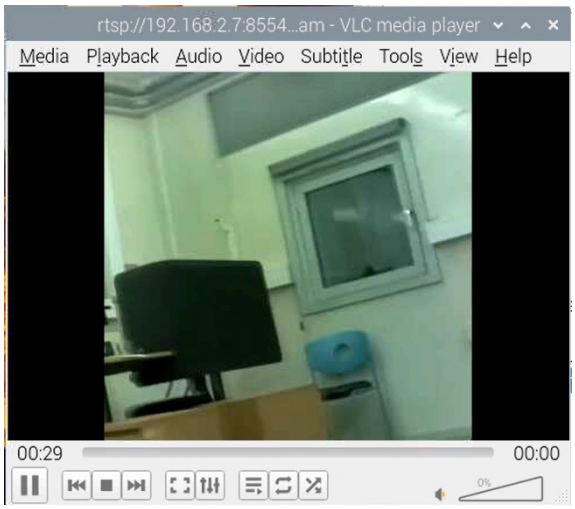


Figure 4.32 - Video stream frame. The frame is complete, allowing the full image to be clearly seen

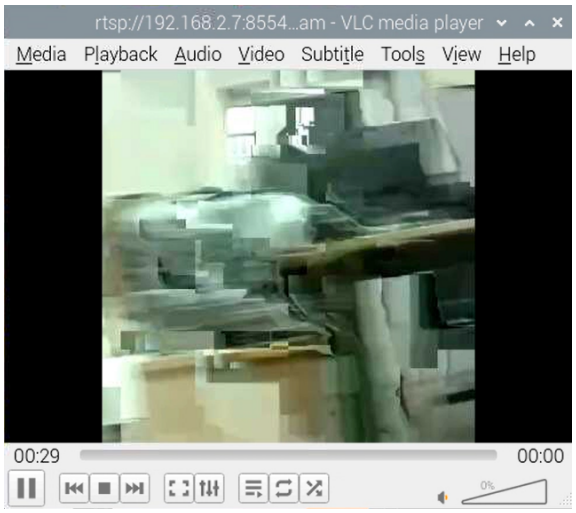


Figure 4.33 - Video stream frame with signals of glitching. This occurs when the video is handling missing data.

4.5.4 TDMA transmissions synchronization

To verify whether synchronization between the nodes was achieved, a node operating in monitor mode was used to observe the network from an external perspective. Monitor mode does not guarantee that all packets are captured, which may lead to missing packets.

The Figure 4.34 displays the synchronization packets sent during each round, covering a total of 14 rounds. The transmission times are wrapped to the period of the round, allowing for the visualization of whether the transmissions are synchronized. It can be confirmed that the nodes successfully reached synchronization and occupied their designated slot zones. For a network with three nodes, the slots are divided as follows, with each slot being 33.33 ms in duration: [0, 33.33, 66.66, 100] ms.

Monitor mode successfully captured the packets from node 7 to the destination (5) but was

unable to capture the responses. To assess the sending of multiple packets within a slot, we performed an *iperf*3 with a UDP connection with a relay node (6). Node 7 sends 100 packets per second. The graph 4.35 shows the amount of packet that node 7 sends out. The nodes are able to keep the synchronization, and we verify that the transmitions are kept within the defined slot. Graph 4.36 illustrates those transmissions without being wrapped.

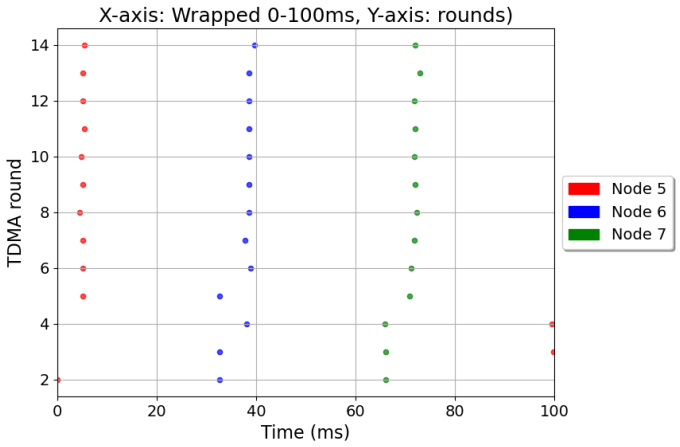


Figure 4.34 - Visualization of synchronization packet transmission. It is possible to verify the synchronization between nodes and see the slot location

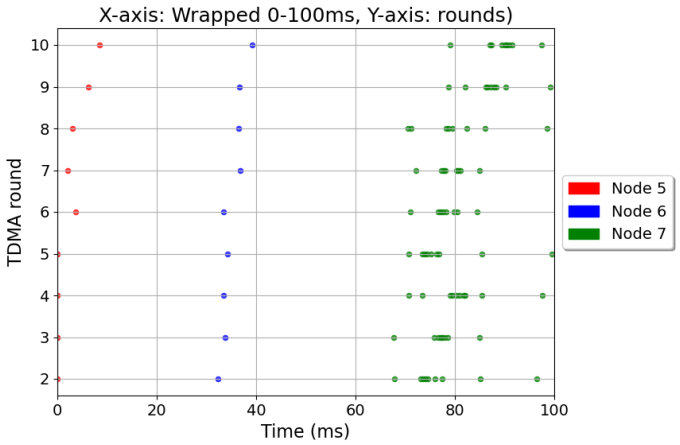


Figure 4.35 - Plotting of the transmission times of multiple UDP packets within the slot of node 7

Finally, we plotted the transmission over the time period when there were four active nodes. Node 7 sends a ping of 6008 bytes to node 10, passing through two relays, nodes 5 and 6. In the graph 4.37, we observed node 10 adjusting its slot when it detected a high flow of packets from node 7. The graph shows only the ping request packets, including the forwarding of packets between relays. The group was able to maintain synchronization while handling the transmission of large ping packets.

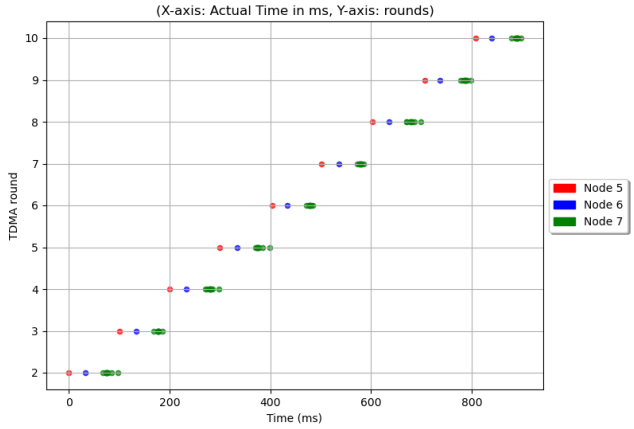


Figure 4.36 - Overview of the transmission times from Figure 4.35

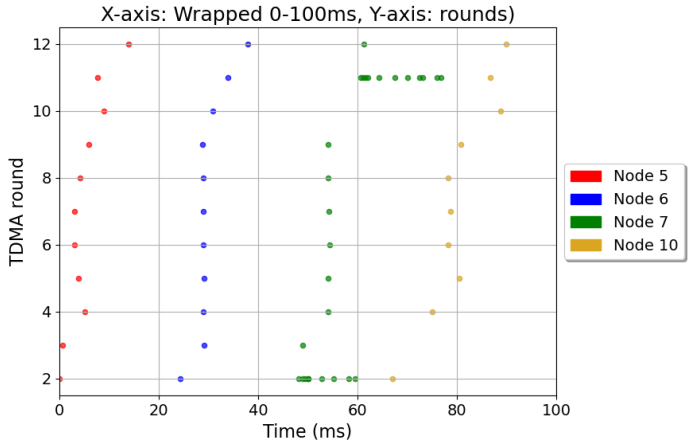


Figure 4.37 - Slot representation with four nodes in the network. Node 7 sends pings of 6008 bytes to node 10 in a 3-hop topology

4.6 Final Remarks

It is important to note that the organization and structuring of the TDMA framework slots, along with the direction of request-reply communications between two nodes, significantly influence performance and outcomes.

In the first example, as illustrated in Figure 4.38, if node 7 initiates the request-reply communication, the RTT for the interaction will depend on the timing of when node 7 sends the packet within its allocated slot. Specifically, if the request is transmitted at the beginning of the slot, the resulting RTT will be longer than that for a request sent toward the end of the slot. This is because, without accounting for interference, when the request is sent later in the slot, node 10 has a shorter waiting period before it can send its response.

In other words, if node 7 transmits its request toward the end of its designated time slot, node 10 is positioned to respond almost immediately after receiving the request, thereby minimizing the

delay before it can initiate its own transmission. Conversely, if the request is sent at the beginning of the slot, node 10 must wait for a longer duration until it can respond, as it has to wait for the current slot (node 7) to conclude before it can utilize its own allocated time for sending the reply.

Figure 4.39 illustrates how the ordering of slots impacts the expected RTT between requests and replies. When node 10 initiates the communication, it must wait until the next round to receive the reply, as the response cannot be sent until the following time slot. This scenario was specifically tested in the experiments. Consequently, the experiments conducted with TDMA, as described in Section 4.5, featured node 10 as the communication initiator. It is crucial to consider these characteristics when analyzing the RTT values associated with the exchange of request and reply messages between two nodes in this network.

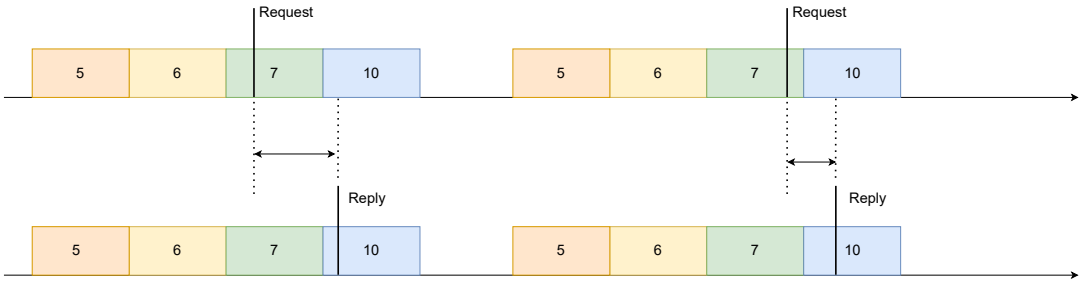


Figure 4.38 - Example of a Request-Reply between node 7 and node 10, respectively. The RTT values vary according to when the request is sent within the slot.

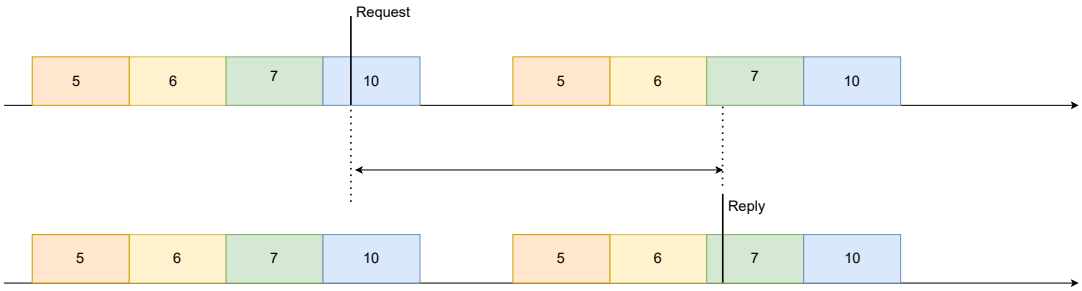


Figure 4.39 - Example of a Request-Reply between node 10 and node 7, respectively. The reply will only be sent in the next round.

4.7 Summary

This chapter presented the experimental components and results of this dissertation. It detailed the testbed used for practical tests and the metrics defined for evaluating communications within the multi-hop network. However, certain limitations were imposed by the testbed because all the nodes were within reach of each other in all the experiments, i.e., they were all in the same collision domain. Thus, breaking links had to be enforced artificially with techniques such as *iptables* for packet dropping.A consequence of this method is that transmissions by nodes that are not linked

(artificially) still interfere with each other, occupying the communication medium and potentially increasing collisions.

Nevertheless, the chapter ended with an extensive series of experiments using the testbed. The experiments validated the capacity for peer-to-peer communication demonstrating reliable video streaming between two nodes using the TDMA structure and with changes in topology and consequently in the routes.

Finally, it is relevant to say that the testbed limitations affect its performance negatively. Thus, in a real operational scenario where links break due to distance or obstacles, the nodes will no longer be all in the same collision domain and we believe the performance will improve, particularly throughput.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this work, we set out to design and implement a routing algorithm tailored for mobile ad-hoc networks operating within the RA-TDMAs+ synchronization framework. The main objective was to establish a reliable routing mechanism that would support efficient peer-to-peer communication in decentralized, dynamic environments without relying on centralized control. By focusing on routing at the data link layer (Layer 2 of the OSI model), we ensured minimal impact on higher-layer protocols and applications, maintaining the integrity of IP-based communication while ensuring adequate network performance. This abstraction, in which all nodes are within the same IP network, allows for simplified application development, where communication can occur without direct involvement in route management.

Through extensive testing in multi-hop networks with changing topologies, we demonstrated that the proposed algorithm efficiently manages network traffic, even as the number of relays increases. Our analysis of key network metrics, including delays and variations in performance, validated the solution in a relevant practical scenario.

Test results without the TDMA framework revealed expected characteristics intrinsic to the multi-hop network. Knowing these characteristics, we could infer the dynamic topology at each instant based on the performance metrics. Packet loss remained relatively low, even with frequent topology changes, and the routing algorithm efficiently updated the routing tables in response to these changes, ensuring swift adaptation to the dynamic topology.

On the other hand, the TDMA framework imposes periodic behavioral characteristics on network transmissions, and this behavior remained consistent across different topologies. As a result, it is not possible to infer the specific topology based on transmission characteristics alone. This demonstrates that the routing and topology are independent of the TDMA framework, and the performance of transmissions is more dependent on the transmission slots rather than the underlying routes. The routes primarily affect the extreme values, such as maximum and minimum performance metrics, due to the number of forwarding retransmissions involved. The average performance is totally dependent on the TDMA configuration and packet size.

In environments where nodes positions change frequently, dynamic ad hoc topologies offer more advantages than using an infrastructure with static relays. The continuous changes in topology help compensate for the occasional longer delays introduced by multiple forwarding, resulting in better overall average performance metrics.

Despite the inherent characteristics of multi-hop mesh networks, which often result in increased delays and reduced throughput, this work demonstrated that it is possible to maintain a working level of communication, even for data-intensive applications such as video streaming. This was achieved both with and without the TDMA framework, indicating that ad-hoc mesh networks can support significant packet exchanges, even in complex network configurations. This finding is particularly significant in scenarios where infrastructure-based networks are unavailable or impractical.

In summary, the findings in this dissertation underscore the effectiveness of the proposed routing algorithm within the context of dynamic ad-hoc mesh networks, particularly when coupled with the RA-TDMAs+ framework. The proposed solution provided robust communication with low packet losses and was validated in a relevant practical testbed with controlled topology that allowed breaking and restoring communication links as desired. The testbed included a mobile robot carrying a camera and transmitting a video stream to another node while moving.

5.2 Future Work

One aspect that could significantly enhance the framework performance is incorporating link quality indicators into the topology matrix. This would ensure a more efficient operation of the Prim algorithm (when creating the spanning tree of the synchronization process) and of the routing process. Moreover, incorporating link quality can facilitate the development of a topology formation strategy that enhances connectivity, allowing the team of robots to maintain a more robust configuration tailored for specific missions. This structured approach would improve the efficiency and effectiveness of mission execution, ensuring that the robotic team remains organized and responsive to dynamic operational requirements.

Another aspect that would impact positively the network performance is the scheduling of packets within each slot based on their routes and packet sizes. This could ensure packets reach their destination before the slot ends and could avoid potential collisions between new packets sent by the source and pervious packets being concurrently forwarded by the relays. The first aspect requires defining a precise latest transmission time in the slot, by the source node. The second aspect can be enforced adding an adequate time interval between consecutive packets sent by the source to give time for each packet to be forwarded to the destination. However, how to enforce these timings in a complex operating system is not trivial and requires further investigation.

It would be interesting to use the developed testbed to validate in practice the study presented in [3]. We could also explore the performance differences between forwarding packets on behalf of the source node during its own slot versus storing the packets and waiting for its own slot to forward them. These approaches impact transmission characteristics and network delays. We

believe that forwarding within the slot of each node may help reduce collisions caused by relay retransmissions, which is harder to enforce with immediate forwarding.

Finally, it would be interesting to explore utilizing traffic control tools to manage transmissions more precisely to ensure packets are sent strictly within their assigned slot, providing a better control over network performance.

References

[1] D. Almeida. Synchronization for Adhoc Multi-hop Networks of Mobile Agents. Master's thesis, FCUP, 2023.

[2] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 16(2):172988141983959, March 2019.

[3] Luis Oliveira, Luis Almeida, and Pedro Lima. Multi-hop routing within TDMA slots for teams of cooperating robots. In *2015 IEEE World Conference on Factory Communication Systems (WFCS)*, pages 1–8, Palma de Mallorca, Spain, May 2015. IEEE.

[4] Luis Oliveira, Luis Almeida, and Daniel Mosse. A Clockless Synchronisation Framework for Cooperating Mobile Robots. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 305–315, Porto, April 2018. IEEE.

[5] Luis Pinto and Luis Almeida. A Robust Approach to TDMA Synchronization in Aerial Networks. *Sensors*, 18(12):4497, December 2018.

[6] L. Ramos Pinto. *Aerial Multi-hop Sensor Networks*. Ph.d. dissertation, FEUP, 2018.

[7] Mohammed M. Alani. *Guide to OSI and TCP/IP Models*. SpringerBriefs in Computer Science. Springer International Publishing, Cham, 2014.

[8] S Sharmila and T Shanthi. A survey on wireless ad hoc network: Issues and implementation. In *2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS)*, pages 1–6, Pudukkottai, India, February 2016. IEEE.

[9] Shihu Xiang and Jun Yang. Performance reliability evaluation for mobile ad hoc networks. *Reliability Engineering & System Safety*, 169:32–39, January 2018.

[10] Naeem Raza, Muhammad Umar Aftab, Muhammad Qasim Akbar, Omair Ashraf, and Muhammad Irfan. Mobile Ad-Hoc Networks Applications and Its Challenges. *Communications and Network*, 08(03):131–136, 2016.

[11] Antonio Guerrieri, Valeria Loscri, Anna Rovella, and Giancarlo Fortino, editors. *Management of Cyber Physical Objects in the Future Internet of Things: Methods, Architectures and Applications*. Internet of Things. Springer International Publishing, Cham, 2016.

[12] L. Oliveira, L. Almeida, and F. Santos. A loose synchronisation protocol for managing rf ranging in mobile ad-hoc networks. In *RoboCup 2011: Robot Soccer World Cup XV*, volume 7416, pages 574–585, 2011.

[13] Dinesh Ramphull, Avinash Mungur, Sheeba Armoogum, and Sameerchand Pudaruth. A Review of Mobile Ad hoc NETwork (MANET) Protocols and their Applications. In *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 204–211, Madurai, India, May 2021. IEEE.

[14] Ahmed M. Eltahlawy, Heba K. Aslan, Eslam G. Abdallah, Mahmoud Said Elsayed, Anca D. Jurcut, and Marianne A. Azer. A Survey on Parameters Affecting MANET Performance. *Electronics*, 12(9):1956, April 2023.

[15] Alex Hinds, Michael Ngulube, Shaoying Zhu, and Hussain Al-Aqrabi. A Review of Routing Protocols for Mobile Ad-Hoc NETworks (MANET). *IJIET*, pages 1–5, 2013.

[16] Ali H. Wheeb and Nadia Adnan Al-jamali. Performance Analysis of OLSR Protocol in Mobile Ad Hoc Networks. *International Journal of Interactive Mobile Technologies (iJIM)*, 16(01):106–119, January 2022.

[17] Fahad Taha AL-Dhief, Naseer Sabri, M.S. Salim, S. Fouad, and S. A. Aljunid. MANET Routing Protocols Evaluation: AODV, DSR and DSDV Perspective. *MATEC Web of Conferences*, 150:06024, 2018.

[18] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24(4):234–244, oct 1994.

[19] Akshai Aggarwal, Savita Gandhi, and Nirbhay Chaubey. PERFORMANCE ANALYSIS OF AODV, DSDV AND DSR IN MANETS. *International Journal of Distributed and Parallel systems*, 2(6):167–177, November 2011.

[20] David B. Johnson and David A. Maltz. *Dynamic Source Routing in Ad Hoc Wireless Networks*, pages 153–181. Springer US, Boston, MA, 1996.

[21] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, USA, 1999. IEEE.

[22] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. In *Proceedings. IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century.*, pages 62–68, Lahore, Pakistan, 2001. IEEE.

Appendix A

Ad Hoc Interface Mode in Linux Systems

This appendix summarizes the possible methods to change the interface mode of a Linux system to ad hoc mode and to setup the network.

A.1 Configuration of Ad Hoc Interfaces

First, verifying that the network device supports ad hoc mode is essential. This can be done by executing the following command, assuming the interface to be used is named *wlan*0: *iw dev wlan0 info*. This command will provide information about the *wlan*0 interface, including its supported modes. This command provides detailed information about the *wlan*0 interface, including its supported modes. The section labeled "Supported interface modes" will display "Ad-Hoc" or "IBSS" if they are listed among the supported options.

To change the network interface mode, it is essential to ensure that no services control the interfaces, as this may prevent modifications. If the only network management service is NetworkManager, it is possible to change the interface mode successfully. However, if services like *dhcpcd* are active, changing the interface mode may not be possible, as they can take control of the network configuration and restrict manual adjustments.

Following the example of the *dhcpcdp*, is it possible to verify if the *dhcpcd* service exists and if it is active by running: *sudo systemctl dhcpcd status*.

Adding the following line to the file "/etc/wpa_supplicant/wpa_supplicant.conf" is recommended: add_scan=2. This setting enables the system to always scan for networks, even when no networks are configured.

A.1.1 Configuration with *dhcpcd* service

If the *dhcpcd* service (or other similar) is active, there are two options: either disable the service and reboot the system or install NetworkManager.

- **Disable** *dhcpcd*: *sudo systemctl dhcpcd disable*, and then reboot. Then, it is possible to change the interface mode;
- **Install NetworkManager**: Installing the NetworkManager and starting the service when necessary or using a start-up script like *rc.local*. To start the service, it is necessary to run: *sudo systemctl NetworkManager start*. The Code Block A.1 installs the NetworkManager service and a GUI icon. This method simplifies the activation of ad hoc mode without diverting resources from other services, allowing it to be enabled as needed. One key advantage is that using the *dhcpcd* service can maintain an infrastructure connection for SSH access. This enables the activation of ad hoc mode on the same interface or another existing interface upon request.

```
1 sudo apt update
2 sudo apt install network-manager network-manager-gnome openvpn openvpn-systemd-
      resolved network-manager-openvpn network-manager-openvpn-gnome -y
```

Code Block A.1 - Installation of the NetworkManager service

After applying and selecting one of these settings, Section A.1.2 provides instructions on how to change the interface mode for the ad hoc network.

A.1.2 Change interface mode to Ad hoc

If there is no active *dhcpcd* service or any other service managing the interfaces restrictively, it is possible to change the interface mode to ad hoc mode.

- **Manually**

Code Block A.3 shows how to manually change the interface to mode ad hoc and the setup of the desired network, following the configurations used in this dissertation.

```
1 sudo ifconfig wlan0 down
2 sudo iwconfig wlan0 mode ad-hoc
3 sudo iwconfig wlan0 channel 1
4 sudo iwconfig wlan0 essid RPi
5 sudo ifconfig wlan0 up
```

Code Block A.2 - Setting up manually the ad hoc network

- **Automatically**

If there is no active service enforcing infrastructure mode on the interface, the device can be automatically started in ad hoc mode with the configured network. The file located at */etc/network/interfaces* needs to be modified to accomplish this (Code Block A.3).

```
auto wlan0
iface wlan0 inet static
    address 192.168.2.X
    wireless-mode ad-hoc
    wireless-essid RPi
    wireless-channel 1
```

Code Block A.3 - Update of the interfaces file. This file can setup the ad hoc network and the static IP (X corresponds to the ID)

Even if there is a service that overrides the *interfaces* file, it is still advisable to edit this file with the content from Code Block A.3, as this configuration allows the interface to have a static IP address of its choosing.