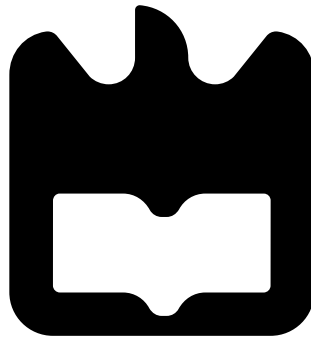




**Kevin Filipe Ganhito
Pinto**

**Mecanismos de Redundância Híbrida para Redes
Wireless Soft Real-Time
Hybrid Redundancy Mechanisms for Soft
Real-Time Wireless Networks**





**Kevin Filipe Ganhito
Pinto**

**Mecanismos de Redundância Híbrida para Redes
Wireless Soft Real-Time
Hybrid Redundancy Mechanisms for Soft
Real-Time Wireless Networks**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Professor Dr. Paulo Pedreiras, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

I dedicate this dissertation to my parents, which supported me in countless ways.

Júri

Presidente

Professor Doutor Pedro Nicolau Faria da Fonseca

Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Vogais

Doutor Paulo Jorge de Campos Bartolomeu

Diretor, Globatronic - Electrónica e Telecomunicações, Lda

Professor Doutor Paulo Bacelar Reis Pedreiras

Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)

Acknowledgment

I would like to express my gratitude to my family, especially to my mother Adélia, who supported me during my whole life and did the impossible to provide me an happy life. I would also like to thank to all my friends, especially Edgar Gonçalves, Daniela Ribeiro, Manuel Roda, Miguel Inocêncio which supported me during all the course and made me a better person. Mihaela Barac for having patience and helping me with the grammar. To all the embedded system laboratory colleagues, which provided me a family-like work environment and kept everyone smiling. I would like to thank to professor Paulo Pedreiras for his patience in supervising me during my long and hard journey in the dissertation and also his motivation.

Palavras Chave

ad-hoc, Ethernet, wireless, IEEE 802.11, soft real-time, mesh network, spatial redundancy, time division, TDMA, Linux, Streaming

Resumo

O streaming de vídeo em ambientes com alta probabilidade de perda de pacotes necessita do uso de mecanismos de redundância. O protocolo WSRT foi criado para melhorar a probabilidade da entrega dos pacotes numa rede mesh multi-hop em meios canais com elevadas perdas usando o protocolo IEEE 802.11.

Wireless Soft Real-Time (WSRT) cria caminhos múltiplos para minimizar nós em comum entre estes, e quando possível, reencaminha a informação por cada um destes, aumentando assim a probabilidade de recepção. Além disto, este sistema usa um mecanismo de TDMA para evitar colisões, melhorar a eficiência energética e largura de banda.

Para alcançar estes objectivos o sistema está decomposto em 3 fases diferentes. Anúncio na qual o sistema irá ter conhecimento de todos os nós. Definição de caminho na qual múltiplos caminhos disjuntos são distributivos e modo de operação na qual o sistema retransmite dados úteis entre dois pontos, a fonte e o consumidor.

O protocolo foi implementado em Linux. Esta dissertação descreve a arquitetura, implementação e um conjunto de resultados experimentais, que validam as características do protocolo, também incluídos.

Keywords

ad-hoc, Ethernet, wireless, IEEE 802.11, soft real-time, mesh network, spatial redundancy, time division, TDMA, Linux, Streaming

Abstract

Video streaming in environments with high probability of packet losses requires the use of redundancy mechanisms. The WSRT protocol was created to improve the packet delivery probability in multi-hop mesh networks with lossy channels by employing IEEE 802.11 protocol.

WSRT creates multiple disjoint paths which minimize common nodes between them, whenever possible, and relays the same data by each one of them, thus increasing the reception probability. Moreover, it also uses a TDMA scheme to avoid collisions and so improve energy and bandwidth utilization.

To achieve these objectives the system is decomposed in 3 different phases, Announcement in which the system will have knowledge of every node in the network. Path Definition in which the multiple disjoint paths are distributed and Running in which the system relays usefull data between two points, source and sink.

The protocol was implemented on Linux. This dissertation describes the architecture, implementation and a set of experimental results, which validate the main protocol features, is also included.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objective	1
1.3	Dissertation organization	2
2	State of the Art	3
2.1	Computer Networking	3
2.1.1	Packet and Circuit Switching	5
2.1.2	Network Models	8
2.1.3	Link Layer	9
2.1.3.1	Multiple Access Link and its protocols	9
2.1.4	Wired and Wireless Communications	10
2.1.4.1	Ethernet	10
2.1.4.2	Institute of Electrical and Electronics Engineers (IEEE) 802.11	11
2.1.4.3	Others wireless communication protocols	12
2.1.5	Routing Protocols	13
2.2	Embedded Systems	13
2.2.1	Linux	14
2.3	Multimedia	14
2.3.1	Video	14
2.3.2	Audio	15
3	System Architecture	17
3.1	Announcement	17
3.2	Path Definition	19
3.3	Running State	19
3.3.1	Time Division Multiplexing	22
4	Implementation	23
4.1	System Implementation	23
4.1.1	Packet Structure	25
4.1.2	Announcement	25
4.1.3	Path Definition	25
4.1.3.1	Best Paths Algorithm	26
4.1.4	Running	27
4.1.4.1	Streaming	29

4.1.4.2	Forward	29
4.1.4.3	Store	29
4.1.4.4	Time Division Multiplexing	30
4.1.5	Transmission and Reception	30
4.2	Protocol Usage	31
4.2.1	Requirements and Setup	32
4.2.2	Tools	32
4.2.2.1	Remote Connection and File Transference	32
4.2.2.2	Network Monitorization	33
5	Results and Analysis	35
5.1	Network Initialization	35
5.2	Running Mode	38
5.2.1	Spatial Redundancy	38
5.2.2	Paths Jitter	39
5.3	Jitter between consecutive nodes	40
5.4	Maximum achievable data rate	40
5.5	Results conclusions	43
6	Conclusion	45
6.1	Future Work	45
A	Protocol Usage Example	49
B	Operating System Configuration Instructions	53
C	Cluster SSH Instructions	57
D	Software Upload Instructions	59
E	MATLAB script for announcing phase results processing	61
F	MATLAB script for path definition phase results processing	63
G	MATLAB script for running mode data processing	65
H	Inter node jitter calculation	69
I	Bandwidth calculation	73

List of Figures

2.1	Alice and Bob conversation as a network protocol analogy.	4
2.2	From top left to bottom right Spatial division, Time division, Frequency division, Code division.	5
2.3	Typical Network topologies.	6
2.4	Packet switching example.	7
2.5	Circuit switching example.	7
2.6	Forwarding analogy using mail.	8
2.7	Open Systems Interconnection (OSI) model and Internet Protocol Stack[1]	8
2.8	Time Division Multiplexing scheme.	10
2.9	IEEE 802.11 evolution. [2]	11
2.10	Ethernet frame.	11
2.11	IEEE 802.11 Architecture	12
2.12	On the left is the original picture. The other is the compressed one.	16
3.1	Protocol use case.	18
3.2	WSRT based model and abstraction layers handling by the devices.	18
3.3	WSRT state machine.	18
3.4	Possible typical topology.	19
3.5	Convergence of the connectivity matrix, by only having 3 nodes in the network. With middle node (2) connected to source and middle node (3).	20
3.6	Forwarding tables of all the network. 3.4.	20
3.7	System normal operation. Based in the example of figure 3.4.	21
3.8	Time division nodes allocation based on the network topology of figure 3.4.	21
4.1	Source node WSRT state machine.	24
4.2	Middle node WSRT state machine.	24
4.3	Sink Node WSRT state machine.	24
4.4	Packet structure.	25
4.5	Announcement flowchart.	26
4.6	Path Definition flowchart.	27
4.7	Best path algorithm flowchart based in common nodes minimization.	28
4.8	Streaming flowchart to generate Streaming packets.	29
4.9	Forward flowchart to retransmit packets.	29
4.10	Store flowchart to provide the user with the received data stream.	30
4.11	Sequential packet insertion.	30
4.12	Transmission and Reception flowchart.	31

4.13	Necessary steps to implement the protocol.	32
5.1	Network implemented in laboratory for experiments,the monitor node captures all the network traffic and is placed about the same distance to every node. .	36
5.2	Announcing Convergence with relation to packet losses.	37
5.3	System Error with independent paths.	38
5.4	Paths jitter.	39
5.5	System performance regarding to the transmission between 2 neighbor nodes.	40
5.6	Delay during multiple packet transmissions in the same slot.	41
5.7	Network data rate and system error rate.	41
B.1	Check for IBSS mode support.	54

List of Tables

2.1	Network categories [3].	6
2.2	Network video streaming transmission rates [4].	15
4.1	Comparison table between paths.	28
5.1	Network convergence time.	36
5.2	Announcing packets loss till convergence.	37
5.3	Network path distribution duration.	37
5.4	Table of packet loss percentage for each path and overall system error.	39
5.5	Time Slots Overlapping	42

Glossary

QoS	Quality of Services
OSI	Open Systems Interconnection
WSRT	Wireless Soft Real-Time
ID	Identification
TDMA	Time Division Multiplex Access
IEEE	Institute of Electrical and Electronics Engineers
TCP	Transmission Control Protocol (TCP)
IP	Internet Protocol
TDM	Time Division Multiplexing
FDM	Frequency Division Multiplexing
HTTP	Hypertext Transfer Protocol
SMTP	Simple Mail Transfer Protocol
UDP	User Datagram Protocol
FEC	Forward Error Correction
CRC	Cyclic Redundancy Check
CAMBADA	Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture
MAC	Medium Access Control
TDMA	Time Division Multiplexing Access
CSMA	Carrier Sense Multiplexing Access
AP	Access Point
IoT	Internet of Things
JPEG	Joint Photographic Experts Group

MPEG	Moving Picture Experts Group
CD	Compact Disk
DVD	Digital Video Disc
FPS	Frames per second
AAC	Advanced Audio Coding
LC	Low Complexity
SSH	Secure Shell
SFTP	Secure Shell (SSH) File Transfer Protocol
BSS	Base Station Subsystem
IBSS	Independent Base Station Subsystem (BSS)
OS	Operating System
AP	Announcing Packets
PP	Path Packets
SP	Streaming Packets

Introduction

1.1 Motivation

The necessity to transmit high amounts of data, like video streaming, between two distinct points in lossy environment, implies the use of a suitable protocol to attain acceptable Quality of Services (QoS) level. An example of a real life problem is video streaming from a lake or a river to land. Water absorbs the electromagnetic radiation generated by the Wi-Fi antennas due to the fact of being a relatively high frequency, 2.4 GHz. The objective of this dissertation is developing an IEEE 802.11 based communication protocol featuring spatial redundancy mechanism to achieve acceptable packet delivery ratio in harsh environments and avoiding data collision. As an additional requirement, the protocol shall be lightweight in order to be deployable on low cost hardware platforms, namely single board computer such as Raspberry Pi and similar platforms.

1.2 Objective

This dissertation aims at creating a lightweight protocol for the IEEE 802.11 standard to transmit data between two distinct points using intermediary nodes to retransmit the information through multiple paths and also in different time slots to avoid packet collisions and increase its delivery rate. Developing such system requires a timely platform and in this case it will be employed a soft real-time system to schedule the transmission of data packets with a reasonable degree of accuracy, providing spatial redundancy and temporal division by assigning paths and transmission time slots to the data. To achieve this objective it is necessary to address the following topics:

- Computer networking: network topologies, network protocols, OSI model, Transmission Control Protocol (TCP) (TCP) and Internet Protocol (IP) model, Ethernet frames structure, routing mechanisms.
- Available options of single boards computers and ways of transmitting data by wireless.
- Study how video and audio are encoded and their typical bit-rate.
- Study Linux programming interface for soft real-time operation.

1.3 Dissertation organization

This dissertation is divided into the following sections:

- **Chapter 1 - Introduction** - Presents the motivation and strategy to solve the problem in transmitting data in lossy environments.
- **Chapter 2 - State of the Art** - Discusses the existing technologies and theoretical background to provide some degree of knowledge to allow the reader understand this dissertation.
- **Chapter 3 - System Architecture** - Represents the global system architecture, including configuration and operational phases such as Announcement, Path Definition and Running.
- **Chapter 4 - Implementation** - Describes how the system operates internally.
- **Chapter 5 - Experimental Results** - presents a set of experimental results that aim at validating the correct operation of the system as well its performance.
- **Chapter 6 - Conclusion** - Summarizes the main conclusions and presents the challenges encountered.

State of the Art

This section contains a collection of data covering a wide range of topics, namely Computer Networking(2.1), Embedded systems(2.2) and Multimedia(2.3), which are closely related with the scope of this work and aim to provide the reader with essential background information.

2.1 Computer Networking

Most of the contents covered in this sections were mostly based using books called Computer Networking - A top-down approach [1], Data Communications and Networking [5], Computer Networks [3] and Data and Computer Communications [6] The information exchange between independent computers through cooper, fiber, microwaves or other medium provides the idea of computer networking [3]. To achieve this information exchange there must be a common language between the computers, the network protocol. It is defined as the exchange of messages and correspondent action or event between two or more communicating entities, hardware or software, like a router, smartphone or a computer. A common analogy is the conversation between two persons. Alice sends a greeting message and Bob understands it and replies by greeting Alice as shown in figure 2.1. By both talking the same language, both can understand each other and this is analogous to the networking protocol [1].

The network is implemented by nodes and links. The role of a node depends on the performed actions. It could either be an end node, to generate and receive data flows, or a router node to forward the data until it reaches its final destination. Links are the entities that provide reliable message exchange between the end nodes. A network is described as a group of nodes¹ and links² to establish the telecommunication between devices through one or many networking protocols depending on the desired actions or events in the nodes [5] [1]. The data exchange, called data flow, through the links can be done in three different ways. Lets say there are 2 devices communicating between each other, A and B. The types of data flow are represented next and were based in [5].

- Simplex

¹Constituted by computers, smart phones, routers

²Entity that permits the transference of reliable messages between nodes.

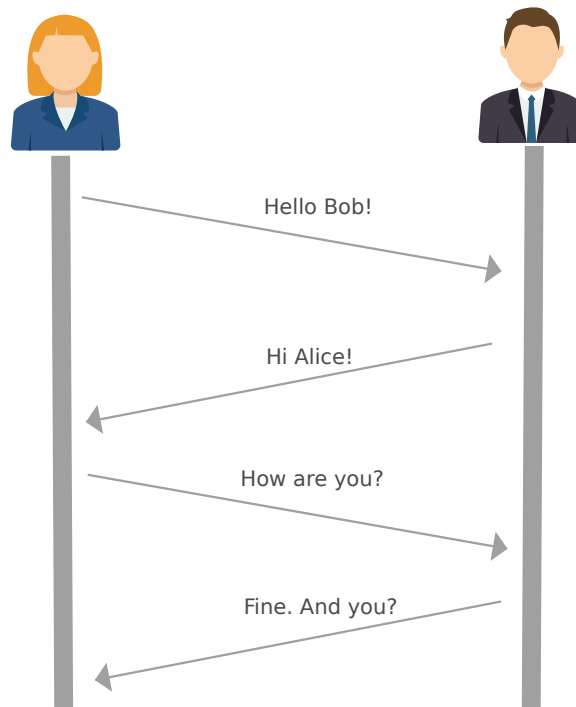


Figure 2.1: Alice and Bob conversation as a network protocol analogy.

The data flow is unidirectional. Device A only sends data to Device B and never the opposite.

- Half-duplex

In this flow mode the transmission/reception can be done by both devices but never at the same time.

- Full-duplex

In full duplex mode both devices can transmit concurrently. In order for this to be possible the link must, e.g: use different frequencies or wires.

Assuming the links are always bidirectional, there are different duplex procedures as described in figure 2.2.

- Spatial Division - uses distinct physical paths to transmit the information. For example one wire to send from A -> B and another to receive from B->A or in wireless networks.
- Time Division - Sends data using slots. Device A and B transmit by turns. A simple example is a road with 2 semaphore and cars. Only one lane can pass each time.
- Frequency Division - implies having a frequency for A and a different frequency for B.
- Code Division - it uses orthogonal codes and shares the same channel.

The arrangement of nodes and links is called a network topology, and there are four basic types: ring, bus, star and mesh. The connections types can be point-to-point or multi point.

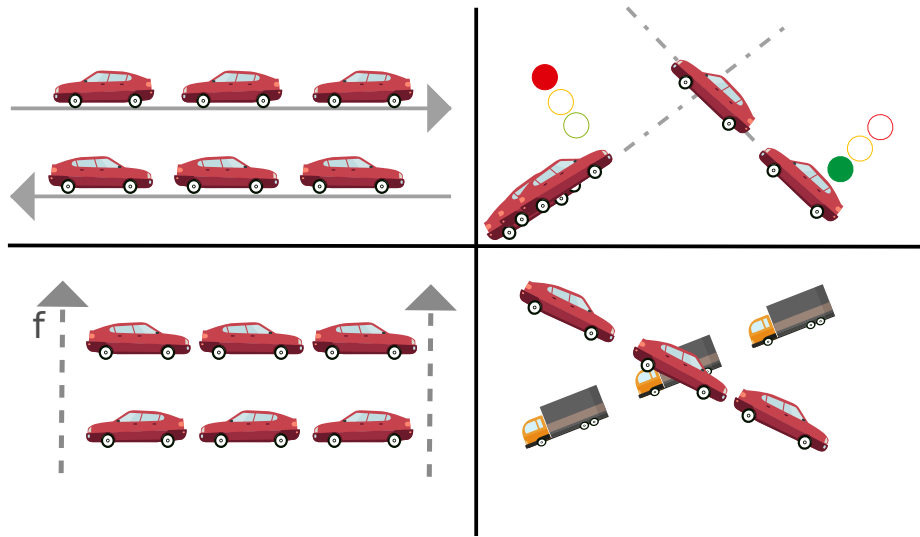


Figure 2.2: From top left to bottom right Spatial division, Time division, Frequency division, Code division.

A point-to-point connection provides a dedicated link between the nodes and so the link capacity is only reserved for that purpose. If the links are shared by more than two nodes it is a multipoint connection and the channel capacity can be shared [5].

Regarding the topologies, this are explained with some detail in [5] in which the described topologies briefly explained below.

In a ring topology each node is only connected to its direct neighbors and due to that it is a point-to-point connection. The data is passed from device to device until the data reaches its final destination.

The bus topology is a little bit different from the ring topology as there is no closed loop and all nodes share the same link, so it is a multipoint connection once all nodes are connected to each other through the bus.

In the star topology all nodes are connected to the same hub. The hub is a single point of failure. If it fails, the whole network fails too. It as a point-to-point link, but the devices don't connect directly between each other.

The mesh topology is a multipoint connection where the links are shared and there are intermediary nodes to forward the data to other nodes.

A network can be separated also into different categories regarding to its size as presented in table 2.1.

2.1.1 Packet and Circuit Switching

Data transfer among nodes can be carried out by two different methods, packet and circuit switching described in [1]. In packet switching method, data is divided into size limited fragments called packets and transmitted through routers independently of each other according to the router forwarding table. It is a connectionless mode once the packets are routed independently. In figure 2.4 packets from computer A to computer B follow distinct paths. Packets are routed independently, meaning that each packet may not be transmitted in the same path.

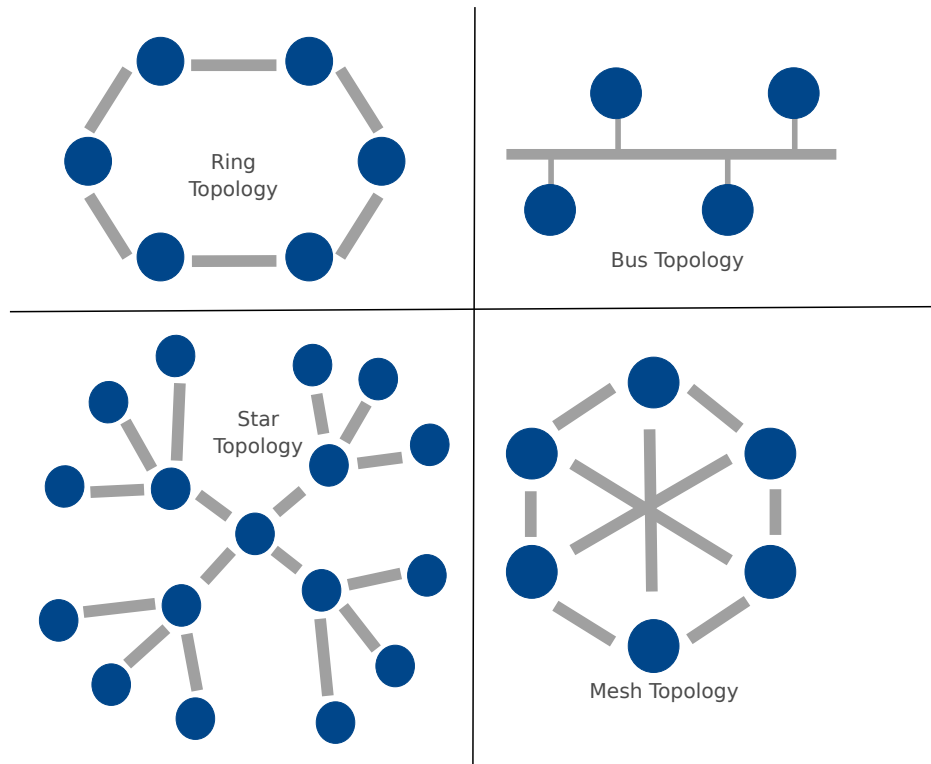


Figure 2.3: Typical Network topologies.

Table 2.1: Network categories [3].

Category	Description
PAN	Personal Area Network - devices communicate in a range of a person. Example: connection between computer peripherals like mouse or keyboard.
LAN	Local Area network - devices communicate in a range of an building like an house. Example: computers connected ti the same wireless router.
MAN	Metropolitan Area Network - devices communicate in a range of a city. Example: Television Network.
WAN	Wide Area Network - devices communicate in a range of a country or continent. Example: Internet.

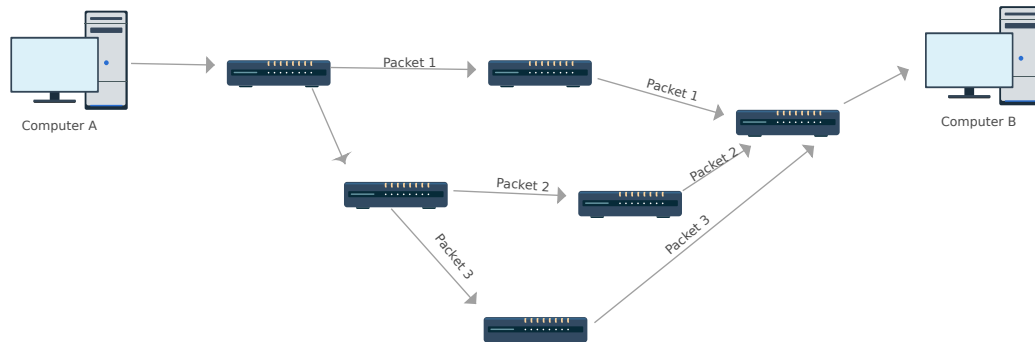


Figure 2.4: Packet switching example.

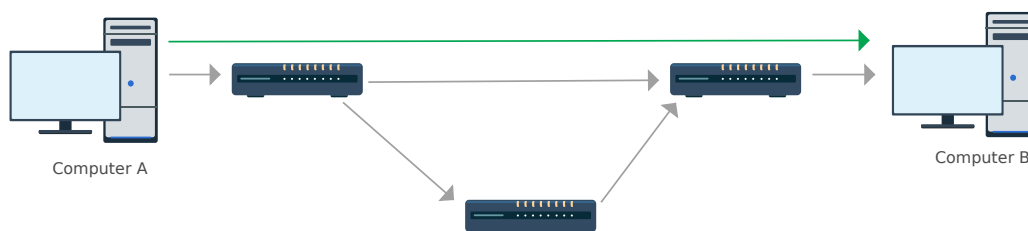


Figure 2.5: Circuit switching example.

Unlike packet switching, circuit switching needs to establish a connection first before transmitting data. The path for a data flow is always the same. This is a connection based transmission. A good example of a circuit switching network is a telephony network. It guarantees a fixed transmission rate until the connection ends, once it has been reserved. In figure 2.5, after the reservation the packets only are transmitted between computer A and B using the green line, which is virtually a direct connection between them.

After the reservation it is necessary to provide some multiplexing mechanism once data can share the same channel and it is necessary to avoid collisions. This can employ Frequency Division Multiplexing (FDM) or Time Division Multiplexing (TDM). Mechanisms which were discussed and represented earlier in figure 2.2.

From a resource efficiency point of view packet switching is more efficient because data packets can be forward through different paths and packets are forwarded independently. In circuit switching if the computer A is not transmitting any data there is a waste of bandwidth for that transmission once it has been previously reserved.

All the router known connections form the forwarding table. This permits the router compare packet destination and determine to which link it should forward it. This table is generated using a routing protocol that will be discussed later. It is important to notice that this routing protocol is one of the many network protocols. A good analogy for the forwarding table is the mail transport between 2 cities. Each city has its own post office and there is a transporter for the mail exchange between the 2 cities and then the city postman delivers the mail to each house of its city. In order for the mail to reach its final destination it is necessary to know to who is addressed to, city A or B at a first level. On a second level it is necessary to have knowledge about the city streets and doors. In each city the post office represents a router, the mail and transporter corresponds to links, while each house is a end node such as

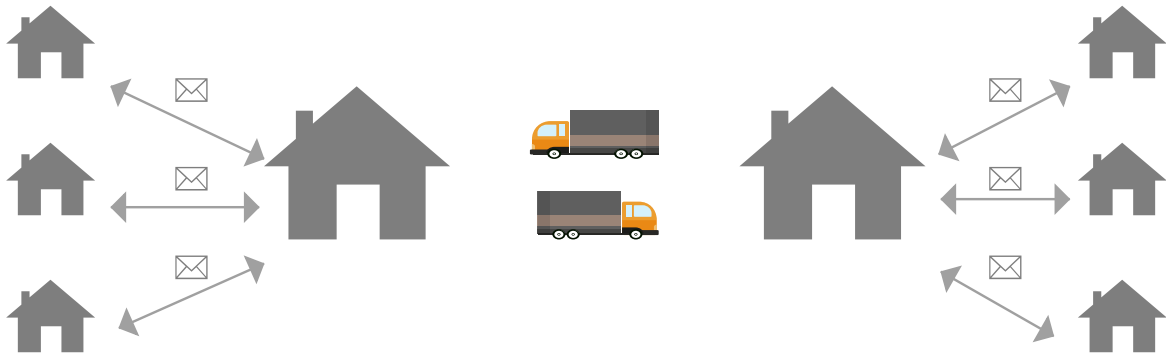


Figure 2.6: Forwarding analogy using mail.

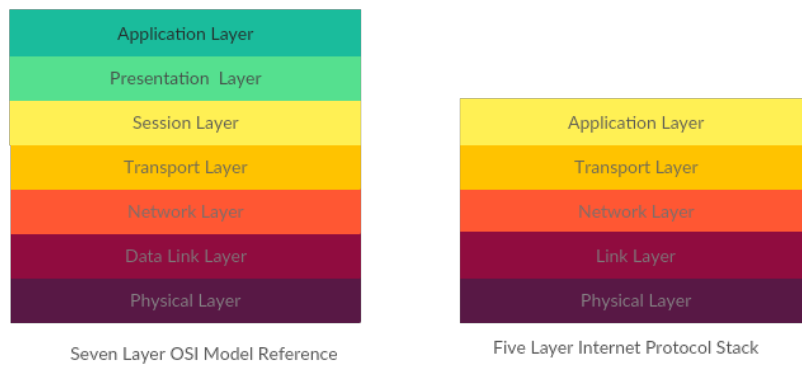


Figure 2.7: OSI model and Internet Protocol Stack[1]

a computer as it is described in figure 2.6.

2.1.2 Network Models

In an heterogeneous network the devices are developed by different entities, and so it is necessary to have a standard to achieve connection between different devices. The most known standards are the OSI and TCP/IP models presented in figure 2.7. Each layer was created to provide an abstraction to perform a well defined function. In this section only the Internet Protocol Stack will be briefly explained, based on Computer Networking - A Top Down Approach [1], once it is the standard used in the project development.

1. **Application Layer** - this layer includes protocols that support user applications, as well some known network services. Example are the Hypertext Transfer Protocol (HTTP)³ and Simple Mail Transfer Protocol (SMTP)⁴.
2. **Transport Layer** - used to convey the application layer message between end nodes

³Request/Transfer of web documents

⁴Transfer of e-mail messages

using TCP⁵ or User Datagram Protocol (UDP)⁶.

3. **Network Layer** - its functions are addressing, rejecting and routing, allowing network packets, known as datagrams, from one node to another eventually across multiple networks. This layer is used by routers to forward the datagrams by using the IP protocol. This layer also provides routing protocols to route packets between source and destination
4. **Link Layer** - A little bit similar to the network layer, the link layer provides the ability to transmit the packets, frames, between a node and its neighbor, but it only operates on the link to which the node is connected to. This layer has some the very important protocols such as Ethernet and IEEE 802.11, that is widely used nowadays and the protocols will be discussed with some detail.
5. **Physical Layer** - moves frames between one network element to another, it converts the bits into electric signals and modulates/demodulates the electrical signals.

2.1.3 Link Layer

As the link layer is very important for this dissertation, it will be explained with more detail. The layer has a Medium Access Control (MAC) to impose rules in frames transmission. Coordinating the frame transmissions among the nodes. The link layer may also provide a reliable delivery service adding an extra overhead to transmit the network layer datagram. This service is usually used in wireless networks where the errors can be corrected locally. Some extra bits are added to check which bits have error and correct it, such as Forward Error Correction (FEC) technique. Cyclic Redundancy Check (CRC) only allows to check for errors.

2.1.3.1 Multiple Access Link and its protocols

Having explained the two types of connections point-to-point or multipoint, if the connection is multipoint all nodes may be able to transmit frames and those could collide at the receiver node and tangle the frames, called multiple access problem. To solve the problems there are a few protocols that can be implemented in the network to avoid the broadcast of two or more nodes at the same instant, called multiple access protocols. To reduce the probability of these protocols different categories exist:

- **Channel partition protocol**

These protocols employ techniques already discussed earlier such as time, frequency, space and code division. Giving the relevance to this dissertation, only one of this mechanisms will be discussed with some detail, Time Division Multiplexing Access (TDMA). TDMA protocol gives to each node a specific time slot for each sending frames as shown in figure 2.8 [3]. The respective slot owner has permission to broadcast data

⁵Connection-oriented protocol that guarantees the delivery of application layer messages by returning acknowledgements to the sending node of which segments arrived to the receiver. It also provides congestion-control mechanisms so that the sending node decreases its data rate to avoid the network overflow and consequently lose packets

⁶Connectionless protocol without any flow control and no congestion control. If a packet is lost during the transmission, source node will not have any acknowledge if it was received.

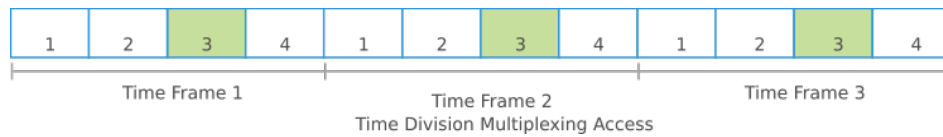


Figure 2.8: Time Division Multiplexing scheme.

frames having guarantees that no other node is transmitting. E.g node 3 has a dedicated slot in the time frame. Synchronization can be distributed or based on a single node, called clock master. A paper called "Self-configuration of an Adaptive TDMA wireless communication protocol for team of mobile robots" [7] dynamically generates the slots according to all active team member and synchronizes their start times based on the previous message transmission and number of nodes on the system. If the channel has uncontrollable load the protocol takes into account also delays the transmission. This is used in Cooperative Autonomous Mobile robots with Advanced Distributed Architecture (CAMBADA) which is a RoboCup Middle-Size League soccer team which shares the same channel with the opposite team and the need of the previously presented TDMA scheme applies.

- **Random Access protocol**

The transmitting node, when there is a collision, retransmits the frame after a random waiting period. The most common are the Carrier Sense Multiplexing Access (CSMA) in which each node listens to the medium to detect if there are any other node transmitting and if not start transmitting [3].

2.1.4 Wired and Wireless Communications

Over the years, multiple ways of transmitting data were created. In wired transmissions, Ethernet is nowadays the standard. In wireless communication IEEE 802.11 [2] variants (different operating frequency, bit rate and function as in figure 2.9) is widely used. There are others wireless protocols such as Zigbee [8], WirelessHART[9], Bluetooth [10] which will be discussed in this section.

2.1.4.1 Ethernet

Ethernet first appearance was in the 80's and it is still widely used in wired LAN [1]. The Ethernet frame structure is represented in figure 2.10 and it is composed by:

- Preamble - it is used to wake up the receiving adapters and to synchronize to the senders clock.
- Destination address - represents the frame final destination. If the address matches the one of the receiver node or a broadcast frame, FF:FF:FF:FF:FF:FF, the link layer passes the data to the network layer. Otherwise the link layer discards the frame.
- Source address - Contains the sender MAC address.
- Ethertype - represents the data chosen protocol, such as IP.

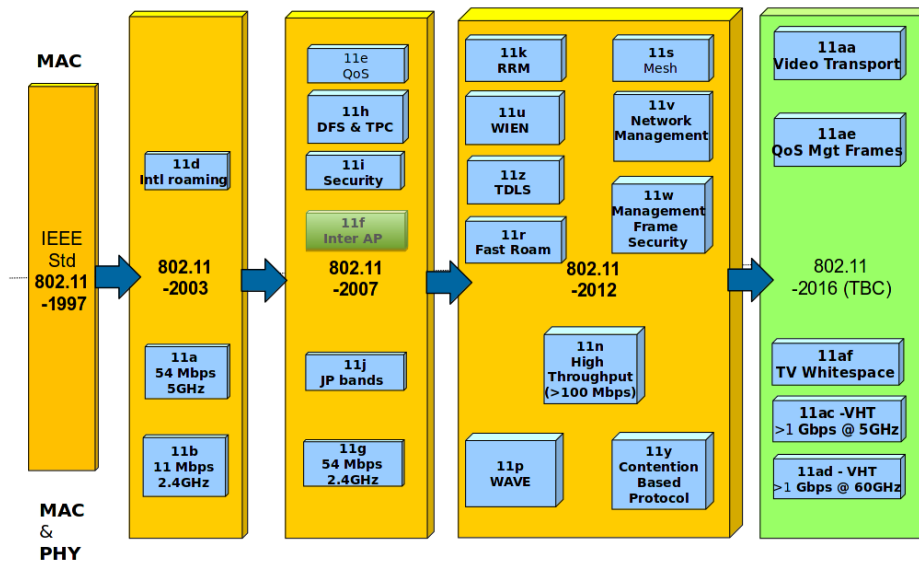


Figure 2.9: IEEE 802.11 evolution. [2]

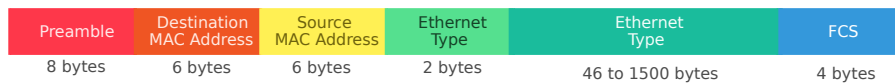


Figure 2.10: Ethernet frame.

- Data - carries the datagram with minimum of 46 bytes and a maximum of 1500 bytes. The datagram is going to be delivered to the network layer.
- Frame Check Sequence - to detect bit errors in the frame.

Over the years Ethernet technologies have been standardized, having different speed, 10 Mbps, 100 Mbps, 10 Gbps over different physical medium, such as coaxial cable, employing Carrier Sense Multiple Access with collision detection, or optical fiber [11].

2.1.4.2 IEEE 802.11

IEEE 802.11 allows secure connections to a network infrastructure at high speeds while being able to keep the connections when changing location while in the network covered area.

It can be used to extend or replace the wired infrastructure, such as the Ethernet previously discussed, creating temporary networks, such as the Ad-hoc network [12].

Figure 2.12 represents the two possible modes to create the network. These modes are:

- Infrastructure mode

In this mode each client is associated with an Access Point (AP) that is connected to other network. The other network could be for example the Internet.

- Ad-hoc network

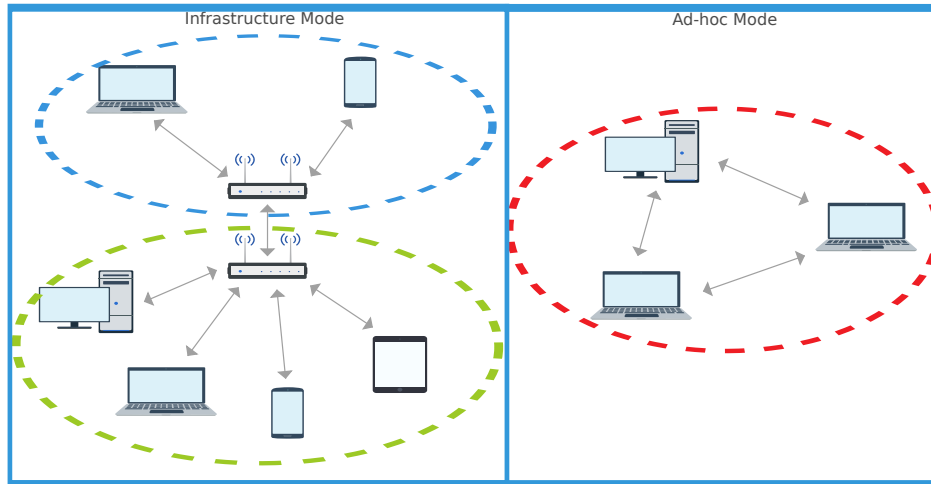


Figure 2.11: IEEE 802.11 Architecture

All devices are associated between them without any central structure such as an AP. The devices can send frames directly between them. This mode is ideal to create a mesh topology.

Both Ethernet and IEEE 802.11 have similar structure, except the link layer in 802.11 is divided into two or more sublayers to make the frames indistinguishable from the network abstraction layer part.

The IEEE 802.11 had many revisions over the years[2], regarding to its speed and frequency. The most common are the 802.11n that works at 2.4/5GHz and provides maximum bit rate between 7.5 Mbit/s to 72.2 Mbit at 2.5 GHz. and 15 Mbit/s to 150 Mbit/s.

2.1.4.3 Others wireless communication protocols

There are many other alternatives to IEEE 802.11. ZigBee protocol typical range is between 10 to 100 meters with a maximum data rate of 250 Kbps [13], although there are some products which have an increased range up, 3200 meters, and data rate of 1 Mbps [14]. It is based in IEEE 802.15.4 suitable for home automation, medical devices and industrial applications. It is reliable because it uses mesh topology, when one device fails the others will keep operating and it is also low power being so ideal for the Internet of Things (IoT)⁷. However its not appropriated for streaming large amounts of data streaming such as video and audio with good quality at the same time [13].

WirelessHART protocol [15] works at 2.4 GHz band and implements a wireless mesh employing TDMA. It is used in measurements and industrial control. This technology was created to satisfy the timing requirements and security in the industrial control. Its network layer provides support to a self-organizing and self-healing mesh network. [16]

Bluetooth was created in order to provide an easy way of connecting short range devices and accessories at low-power. It is composed by a master node that can handle up to 7 connections, known as piconet. And the interconnected piconets are called scatternet [10]. Bluetooth had many upgrades over the years in which allowed the consumption decrease and

⁷Connect everything to the Internet to provide data to monitor, control and understand the environment

range increase of the bluetooth devices. Bluetooth 5 greatly increases the range, speed and message broadcasting when compared with previous versions being ideal for IoT [10].

2.1.5 Routing Protocols

When a packet arrives to a router, the router determines the link to which will retransmit to reach the final destination. To be able to do this, a routing algorithm is necessary to populate the forwarding table. This mechanism is implemented at the OSI network layer [1].

The purpose of employing a routing algorithm is to populate the forwarding tables. Each routing algorithm takes into account possible network problems. Its purpose is to find a "good" path which satisfies the routing protocol conditions. These conditions can be the shortest paths, having minimum number of hops, or packet loss, or signal quality [1]. In the routing algorithm a graph is generated with a set of nodes and edges. Each edge represents the link between nodes and it has a cost. The routing algorithm can choose the least-cost path, by having into consideration all the sum of all possible path edges cost and chose the minimum cost. If all edges have the same cost, the algorithm chooses the shortest path [1].

A routing algorithm can be generated in a global or decentralized manner. The global algorithm computes the least-cost path between source and destination by using all the acquired data. The algorithm can run in a centralized location, where one node calculates the least-cost path and distributes it to the other nodes, or decentralized, where all nodes calculate the least-cost paths. In the decentralized manner the least-cost path is generated without each node having a complete connectivity table, from which can receive data. This iterative process requires the information exchange with its neighbor nodes [1].

Each node forwarding table can be statically or dynamically populated depending if it is the user whose inserts the forwarding table or a routing algorithm, respectively.

There are multiple routing algorithms which can be employed in the network, these are detailed in Computer Networking - A top down approach [1] book and briefly explained below:

- **Link-State (LS) Routing Algorithm** - It a global algorithm where network topology and links costs are known. This is obtained by having every node broadcasting link-state packets and costs of it, and also its attached links. If each node has the same connectivity matrix with the same costs, can compute locally the Link-State algorithm generating the same path in every single node.
- **Distance-Vector (DV) Routing Algorithm** - It is a distributed algorithm that asynchronously uses the received information to calculate the link costs.

2.2 Embedded Systems

Embedded systems is a combination between hardware and software that uses electrical, mechanical and chemical components designed for a specific function. It is widely used in routers, switches, smartphones...

The market is full of embedded solutions for the most different needs, such as the Raspberry Pi [17] or ODROID [18]. Raspberry Pi is an affordable small computer that can be programmed and connected to the Internet. There are a large variety of models with

single/multi-core⁸ and some of them also have WiFi and Bluetooth 4.0 capabilities [17]. These small computers also provide an USB interface. Raspberry Pi supports a big range of operating systems such as Linux based ones, RTOS (Real Time Operating System) and even a special version of Windows for IoT. Raspberry Pi Zero is one of the low cost and affordable computer. It has a single-core processor, and it only has one USB interface which is directly connected to its processor. This capability provides On-The-Go connection and makes possible to program it by a virtual Ethernet interface.

2.2.1 Linux

Nowadays the Linux kernel is supported in almost every processor architecture such as x86, ARM and others. Linux kernel is a clone of the UNIX operating system written by Linus Torvalds. It aims for the portable operating system interface (POSIX) to develop software for different computer architectures. Linux is an open-source operating system (OS) and so ideal to develop application without paying royalties to use the OS. It allows multitasking, virtual memory⁹, shared libraries, networking using IPv4 and IPv6 [19].

Since Linux kernel 2.6, Linux was able to provide soft real-time to its applications natively. Tasks which allow for the Operating System (OS) to occasionally miss deadlines without having a catastrophic impact results is called soft real time. This soft real time is achieved by employing a patch to Linux kernel called PREEMPT_RT. PREEMPT_RT has the goal in minimizing the non-preemptible kernel code and minimizing the changes to provide this functionality without rewriting the whole kernel [20]. Besides soft real-time there is also hard real time in which deadline miss isn't allowed once it could present a major problem for the whole system, for example if a pacemaker misses its deadline it could be catastrophic. Hard real-time can be achieved by patching the kernel with Xenomai [21].

2.3 Multimedia

Video and audio generate lots of data, creating the need of being compressed in order to provide low space occupation and decrease the network bandwidth usage. Data compression imposes one of the biggest challenges in technology as decreasing the data size may generate losses. The compression can be of 2 types:

- Lossless - allows the reconstruction of data as it was before compression. This method can restore data without losses.
- Lossy - with this scheme already compressed data cannot be fully recovered when decompressed.

2.3.1 Video

Video can generate large amounts of data and compression is crucial to minimize the impact when transmitting it in the network. There are 2 basic compression formats:

⁸Having a processor with more than one cores makes the system distribute the load increasing processing speed.

⁹It is a technique in mapping the memory address used by the program, this allows the program being mapped in different RAM locations and hard drive at the same time.

Table 2.2: Network video streaming transmission rates [4].

Quality	Resolution	Video Bitrate
240p	426x240	300 - 700 Kbps
360p	640x360	400 - 1,000 Kbps
480p	854x480	500 - 2,000 Kbps
720p	1280x720	1,500 - 4,000 Kbps
1080p	1920x1080	3,000 - 6,000 Kbps

- Joint Photographic Experts Group (JPEG)

It is a picture compression format widely used today, because it is easy use it. It has a very high compression ratio in exchange of obtaining pictures with degraded quality. There are many JPEG flavours, one of them called Motion JPEG used in digital video sequences using groups of JPEG pictures [22].

- Moving Picture Experts Group (MPEG)

MPEG format is dedicated to digital video sequences it includes many variants such as:

- MPEG-1 compression method using in video Compact Disk (CD) and online video. Offers high compatibility to computers and Digital Video Disc (DVD) players [23].
- MPEG-2 is an upgrade to MPEG-1. Handles higher and larger resolution moving pictures with the disadvantage of increasing the bandwidth usage. It is the DVD standard and used in broadcasting and cable distributed systems [23].
- MPEG-4 is standard for Internet, broadcast, and media storage. Supports progressive scan and interlaced video [23].
- H.264 is designed to provide flexibility to a wide range of applications having latency or bit-rate dependencies. For example: telecommunication services want low-latency to keep a viable conversation or entertainment where can have high latency but lower bit-rate.

Youtube service provides the H.264 compression method for the user to uploads videos in different resolutions, reaching a maximum of 60 Frames per second (FPS) presented in Table 2.2.

2.3.2 Audio

It is crucial for audio to have a small footprint in the transmission, once when transmitting video, audio should also be transmitted. In order to also have a balance in bandwidth consumption audio should also be compressed. There are numerous ways of encoding audio. The most common one in video websites such as Youtube, is the Advanced Audio Coding (AAC) format with low complexity. It was standardized in 1994 as part of MPEG-2. There are different flavors for AAC compression such as Low Complexity (LC) used in Youtube, high efficiency for low bit rates used in Apple iTunes and others, high quality that provides scalable and lossless audio compression [24].

AAC-LC used in Youtube provides audio quality indistinguishable from the original, despite the fact of the original and the coded signal not being mathematically equivalent. Typical



Figure 2.12: On the left is the original picture. The other is the compressed one.

bit-rates are between 128 kbps and 320 kbps depending if it is stereo¹⁰ or 5.1 multichannel signals. AAC-LC also provides a mechanisms to store the audio metadata [4]¹¹.

¹⁰Method to create the illusion of multi direction audio

¹¹Represents the creation data and filename

System Architecture

With the objective of creating a lightweight protocol to handle video and audio streaming between two nodes using a mesh network with high losses due to environmental conditions, the WSRT protocol was created. This protocol is to be deployed in wireless scenarios, where all nodes share the same medium, air. One specific example to employ this protocol is in wet environments, once water absorbs high frequency electromagnetic radiation such as 2.4 GHz used in IEEE 802.11, thus causing high packet losses. To minimize this problem, spatial redundancy will be implemented to provide multiple paths for transmissions, thus increasing the probability of packets reaching its destination. Moreover, a TDMA scheme is also included, to avoid collisions and permit wider bandwidth efficiency.

Each network node may have one of three functions, either is source, a middle or a sink node. The source node has the function of initializing the network by defining the best paths for the data to travel and stream data. Middle nodes are required to forward packets, through the defined paths, until data reaches the sink node, where it will be absorbed and stored.

Figure 3.1 depicts the functionalities the protocol must satisfy. The streamer (source node) can send its data only when the network is initialized. This network initialization requires knowing à priori all the available network connections. The consumer (sink node) accepts the data stream to be later used.

WSRT protocol is located within the network layer of a modified OSI model presented in the abstraction layers of figure 3.2. This system architecture is centralized. The source node responsible in changing the network state machine, represented in figure 3.3, and generate the forwarding tables for every single node in the network. The system architecture is divided into three states announcement, path definition and running. These will be covered in next sections as well the state machine, in figure 3.3, having into consideration an example network topology presented in figure 3.4.

3.1 Announcement

During the Announcement state it is acquired knowledge about all nodes in the network. Each node should announce itself while also sharing all known connections from other nodes as well its own direct connections. When other nodes receive this information they update their local connectivity matrix.

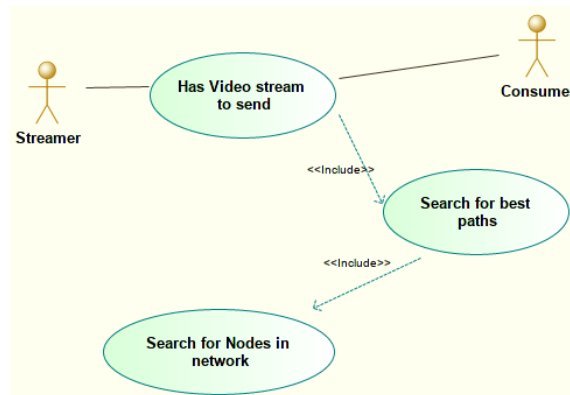


Figure 3.1: Protocol use case.

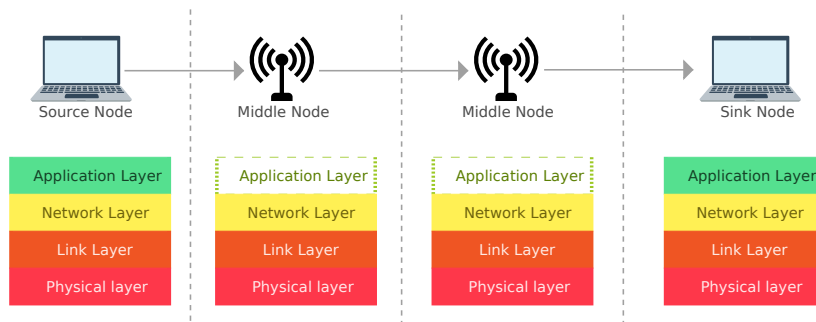


Figure 3.2: WSRT based model and abstraction layers handling by the devices.

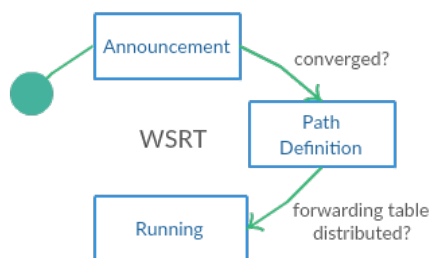


Figure 3.3: WSRT state machine.

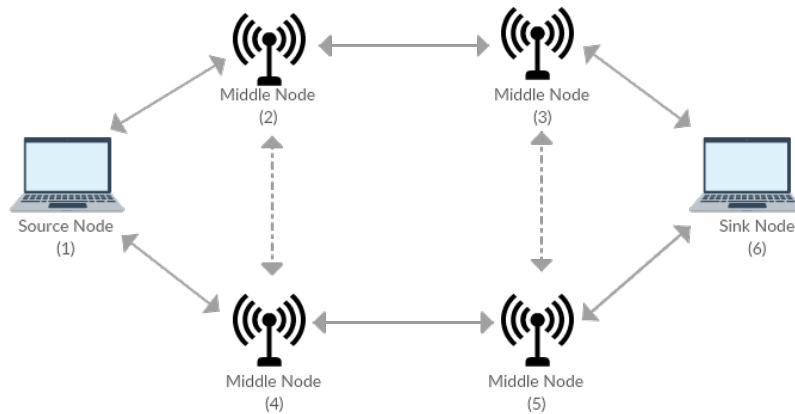


Figure 3.4: Possible typical topology.

Figure 3.5 represents how the connectivity matrix can be generated. Dotted arrows represent the matrix update event, and underline numbers are the new inserted node IDs into the connectivity matrix. The first element of each line represents the node who owns the remaining elements in the same line. These elements are the known connections to the node.

In step 1, the source node starts transmitting its own connectivity matrix. At step 2 the middle node (2) updates its local connectivity table and broadcasts the updated matrix. Step 3 the source and middle nodes (3) join the received matrix to their own local matrix. In step 4 node 1 transmits and node 2 updates its matrix. The same process is repeated through step 4 to 7 until all the nodes connectivity matrix are exactly the same. When that happens the network has converged. When network convergence occurs this state is then concluded and the system can transit to the next state (Path Definition).

3.2 Path Definition

Path definition is the state in which the most favorable paths are distributed through the network. The source node generates these paths by employing an algorithm which creates multiple paths to communicate between source node and sink node through the middle nodes. This algorithm must choose the paths and minimize the common nodes between these paths to make sure the data will be transmitted in a distributed way by avoiding the same data being lost in the same path. The distribution is performed by broadcasting the table in figure 3.6, in which the left column represents the nodes that are allowed to forward data by the nodes on the right column. This table was based in the example network provided before. In this example two virtual circuits are made using paths defined by the source node, through nodes 1, 2, 3, 6 and 1, 4, 5, 6 providing spatial redundancy by two distinct paths. When all nodes get informed of the forwarding tables, the system can move to the next state, in which the actual communication take place.

3.3 Running State

When the system enters in the last state, running, it provides a reliable communication by having data being sent between source and sink node, and forwarded by using the forwarding

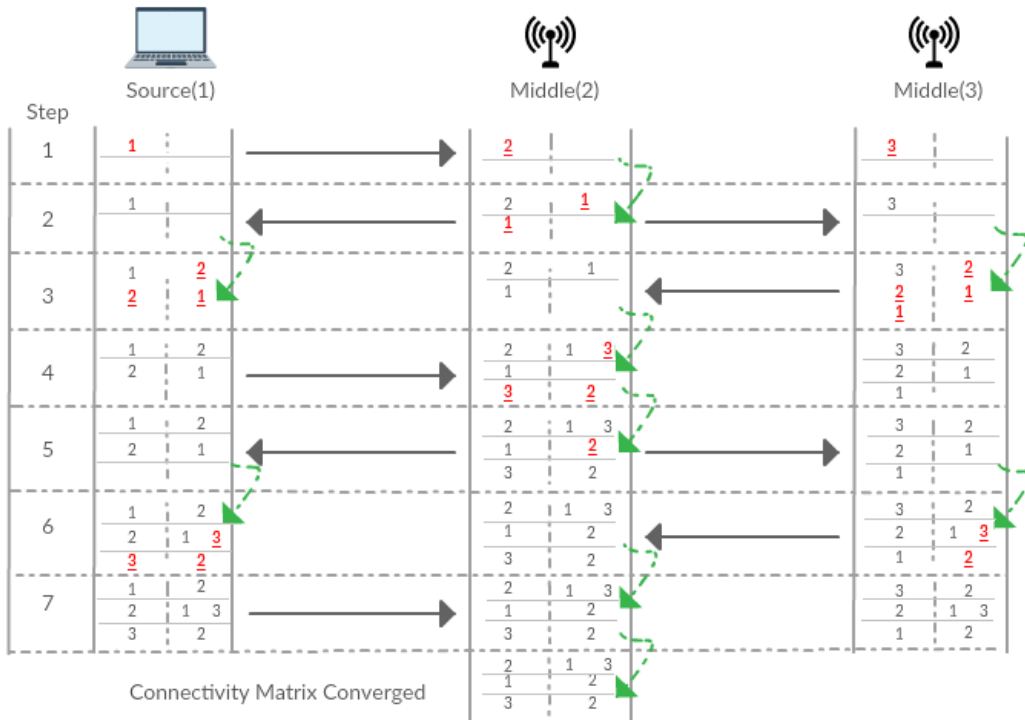


Figure 3.5: Convergence of the connectivity matrix, by only having 3 nodes in the network. With middle node (2) connected to source and middle node (3).

Forwarding Table	
Local ID	Allowed Con ID
1	-----
2	1
3	2
4	1
5	4
6	3 5

Figure 3.6: Forwarding tables of all the network. 3.4.

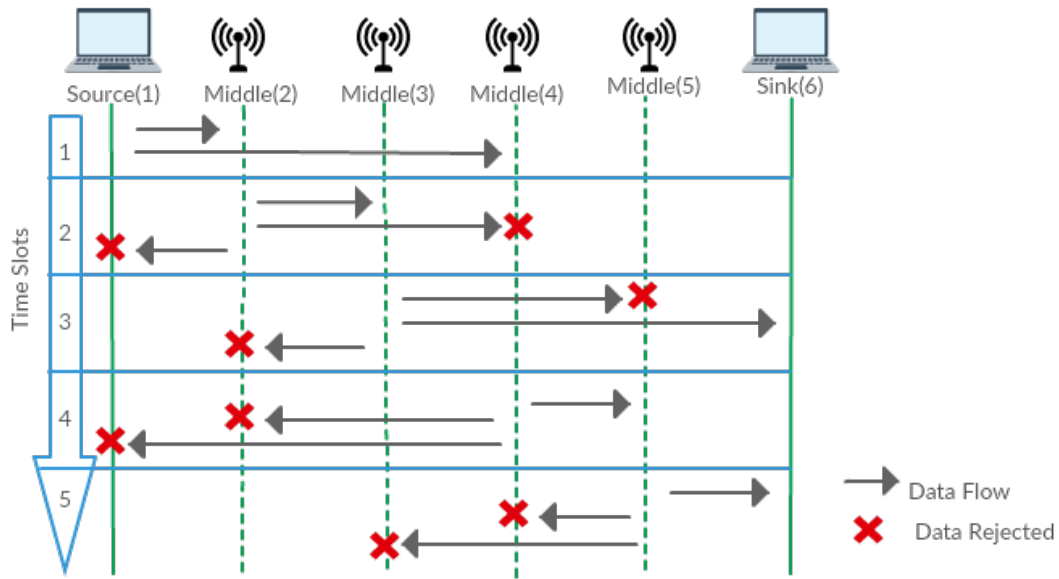


Figure 3.7: System normal operation. Based in the example of figure 3.4.

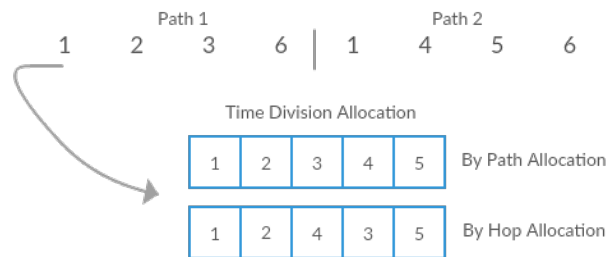


Figure 3.8: Time division nodes allocation based on the network topology of figure 3.4.

table over redundant paths. After the previous network initialization, all nodes have an allowed connection matrix which will allow them deciding which packets they should forward and which ones to discard. Figure 3.7 represents the system normal operation mode, which is based on the example network provided in figure 3.4. During transmissions packets may be rejected, this rejection is represented in the figure with a red cross. When each node receives data which is supposed to forward, waits for its time slot and broadcasts it. The source node broadcasts data, middle nodes 2 and 4 accept it and wait for their slot. The second slot of the frame belongs to node 2, thus it will broadcast the data and nodes 3, 4 and 1 receive it, but only node 3 accepts it since node 2 is declared at its own forwarding table. After that, node 3 will broadcast in time slot 3 and nodes 2, 5 and 6 receive it, but only node 6 accepts it. The same will happen with node 4 and 5. If there are losses in the first path, the existence second path increases the chances of node 6 receiving it. The sink node (6) may receive duplicated data in such case it will discard te duplicate. The sink node (6) also can receive out of order packets, in which the sink will be able to order them.

3.3.1 Time Division Multiplexing

For the system to be able to avoid collisions, it is used a TDMA scheme during the running state. The slot allocation algorithm takes into account the previous forwarding tables and performs the slot allocation. Time frame is created using slots, which is the travel duration of data between source and sink node. The node placement into the frame can be performed into distinct ways, either can be allocation by hop or by path. On allocation by hop the slots are filled by one node of every path sequentially, until there are no nodes left to allocate. On the allocation by path, consecutive time slots are filled with the whole path, until no paths are left. Using the example provided, two allocation modes are represented in figure 3.8. After the source node defines the optimal paths, the allocation takes place and chooses one of the two possibles options. Each node will synchronize itself with the node which is allowed to receive. For example, in allocation by path, node 3 will forward data after receiving it from node 2. By using this mechanism collisions are avoided when nodes broadcast their information, once they schedule transmissions between them.

Implementation

WSRT was developed to be executed in low cost embedded platforms which may not be able to satisfy very precise time constraints. The platform used to deploy this protocol was a Raspberry Pi running Arch Linux.

The system implementation is presented in section 4.1 and the usage of the previous implementation in section 4.2.

4.1 System Implementation

The WSRT protocol has a centralized architecture, with the source node being the one responsible for the overall management. The node which initializes the network and changes all other systems states at network convergence. Middle and Sink nodes are subject to the source node's current state and thus there will be distinct working modes and consequently different state machines.

WSRT protocol has its own packet structure to allow the information exchange between nodes through the different protocol operation modes, Announcing, Path Definition and Running, as described in chapter 3. This structure is presented in section 4.1.1.

The source, middle and sink nodes state machines are represented in figures 4.1, 4.2 and 4.3. All these state machines have the same implementation regarding to the Announcement and Path Definition state. The differences between node types lies in the state transitions and also the last state.

When the source node is started, Announcing Packets (AP) are sent. The AP reception event causes the middle and sink node transit to the Announcement state initiating the AP transmissions. During this process, information about the nodes connectivity matrix is exchanged between all the network nodes, as described previously in section 3.1, and the AP generation is presented in section 4.1.2.

When the convergence event occurs, the source node executes the best paths algorithm, presented in section 4.1.3.1, transiting then to the Path Definition state where Path Packets (PP) will be generated, as described in section 4.1.3, causing the middle and sink node move to the Path Definition state and updating their forwarding table. During this state all nodes will be listening and transmitting PP.

After distribution, source node transits to the Streaming state and runs the TDMA slot allocation, previously presented in section 3.3.1, which will start sending Streaming Packets (SP) and consequently causing transition and execution of the TDMA slot allocation on the middle and sink nodes to the Forward and Store states respectively.

These previous states, Streaming, Forward and Store are sub-states of the Running state, discussed in section 3.3 and implemented in section 4.1.4.

The transmission and reception of the previously generated packets are implemented in section 4.1.5 and are responsible for the send and receive events of the presented state machines.

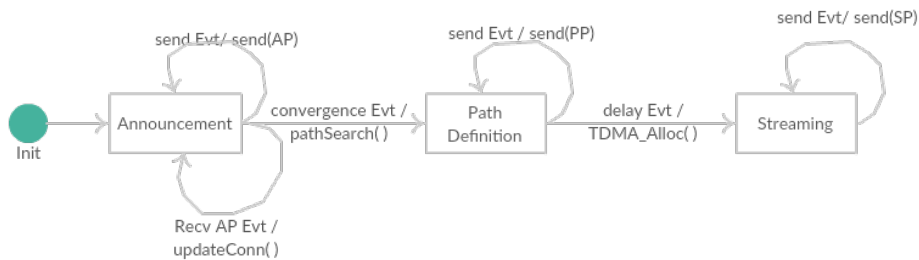


Figure 4.1: Source node WSRT state machine.

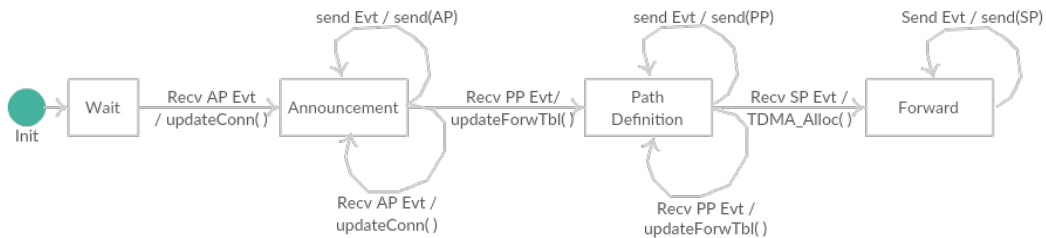


Figure 4.2: Middle node WSRT state machine.

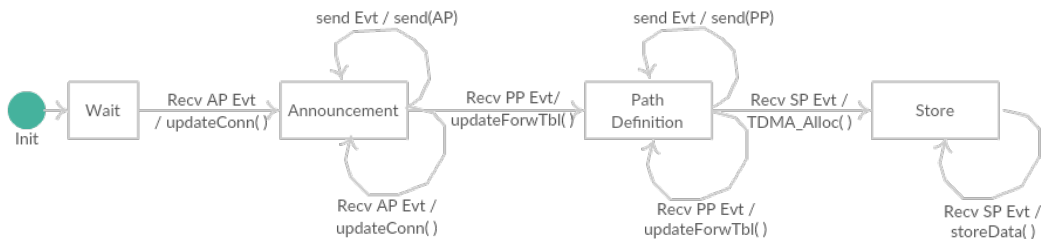


Figure 4.3: Sink Node WSRT state machine.



Figure 4.4: Packet structure.

4.1.1 Packet Structure

To be lightweight, avoiding unnecessary headers, in IEEE 802.11 frame an Ethernet based frame represented in figure 4.4 was used, which only has Destination MAC, Source MAC address and Ethernet Type fields. To properly transmit the data between source and sink nodes, it is necessary to define a packet structure with control fields.

- **Mode** - Defines the current state of the network protocol. It can be of three types: Announcement, Path Definition, Running.
- **Sequence Number** - Represents the packet sequence number which is being forwarded. This is used to eliminate duplicates and sort the packets at sink node.
- **Node Identification (ID)** - sender ID used to retransmit or discard packets according to the sending node forwarding table
- **Payload** - Protocol application data.

4.1.2 Announcement

Announcement state is equal on every node, either it is a source, middle or sink node. This state has the function of providing knowledge to the whole network about all the available connections between the nodes. The transmission and reception flowchart of the AP is represented in figure 4.5.

At the AP reception, the system waits until it has the received AP packet, then the packet payload is searched for tables with possible connections. If there are any tables, the local connectivity matrix will be updated and the search continues until the packet has no information left to be processed. If there are no tables left, the packet is discarded and the system waits for more tables. At some point the connectivity matrix will not have any new information being added, and thus the network has converged.

Regarding to the transmission of AP, the system will wait for a timer event and if the actual system state is to create an AP, the system will transform the local connectivity matrix into tables, copy it into the packet payload and insert it in a waiting queue to be sent as described in section 4.1.5.

The AP send events are achieved by setting a timer. When the timer expires, the correspondent packet is sent. The timer intervals are composed by a base time contained with a random counterpart. This technique is used to avoid probabilistically consecutive collisions among nodes, when they transmit at the same time.

4.1.3 Path Definition

Path definition establishes the paths for application data flows by creating a forwarding table for every active node in the network. These tables are transmitted and retransmitted to every other node until they are distributed by all the participating nodes at the last stage.

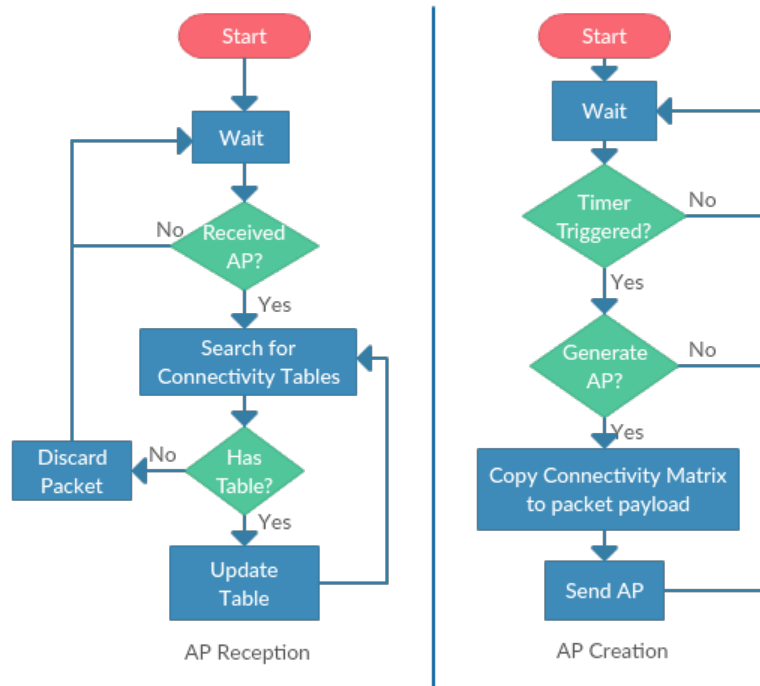


Figure 4.5: Announcement flowchart.

The flowchart in figure 4.6 is similar to the Announcement, except the search is done for forwarding tables. At reception, the system awaits for a PP and searches it for the forwarding table. Then the node updates its internal forwarding table data with the received ones and forwards it to allow the dissemination through the whole network.

To create a PP, the system waits for a trigger to generate it, and if the source, middle or sink node are at the Path Definition state, the system will copy the local node forwarding table and transmit it by putting the PP in a waiting queue to be sent, will presented in section 4.1.5.

4.1.3.1 Best Paths Algorithm

To provide the maximum redundancy possible, the best paths algorithm was created. This algorithm generates paths by minimizing common nodes, trying to avoid the same data being forwarded by the same node. Figure 4.7 represents the process to obtain the best paths.

First it is necessary to generate all possible paths. Then a table of similar paths is created. This is done by comparing every single path node with each other and counting the common nodes. After that, the paths chosen are the ones with a minimum of nodes in common.

For the network example presented in figure 3.4 there are 8 possible paths. The comparison between every path is represented in table 4.1. For example, path 0 is compared up against every other path. When compared with path 1 it detects 4 common nodes. When compared to path 2, 3, 4, 5, 6 and 7 there are 4, 3, 4, 4, 3 and 2 common nodes respectively. This process is repeated until all comparisons between paths are processed.

- Path 0 : 1 2 3 6

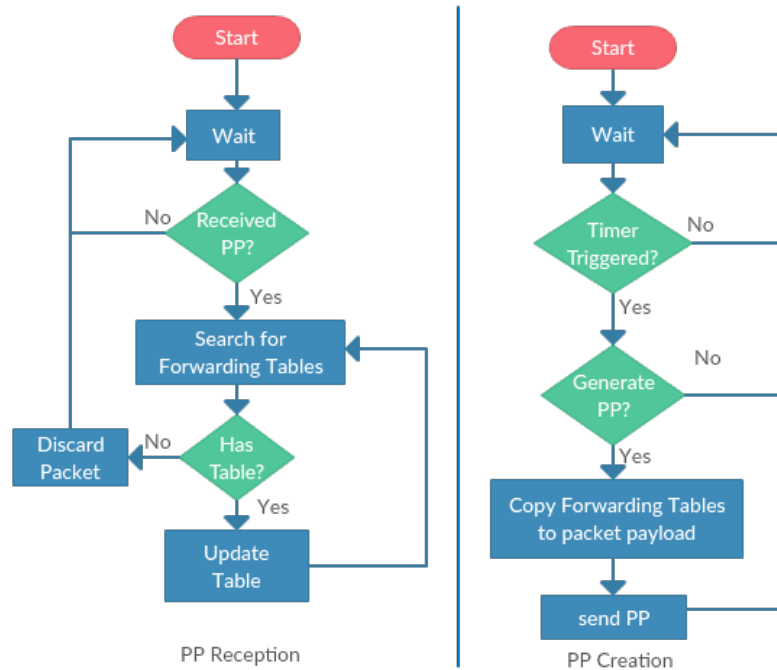


Figure 4.6: Path Definition flowchart.

- Path 1 : 1 2 3 5 6
- Path 2 : 1 2 4 5 3 6
- Path 3 : 1 2 4 5 6
- Path 4 : 1 4 2 3 6
- Path 5 : 1 4 2 3 5 6
- Path 6 : 1 4 5 3 6
- Path 7 : 1 4 5 6

4.1.4 Running

The Running state implementation is more complex than the previous states, since it depends on the node type. The source node Streaming state implementation is explained in section 4.1.4.1, middle node Forward state in section 4.1.4.2 and sink node Store state in section 4.1.4.3. Both states work according to the TDMA allocation previously executed in which transmissions are schedule depending the received message and slot position in the TDMA frame. This is presented in 4.1.4.4. Middle and Store node have a mechanism to discard duplicated packets. This is done by having a fixed sized table to record the previously received SP sequence number. If the packet number is presented in the table, packet is discarded, otherwise it is accepted and its sequence number will substitute the oldest packet in the table.

Table 4.1: Comparison table between paths.

Path	0	1	2	3	4	5	6	7
0	-	4	4	3	4	4	3	2
1	-	-	5	4	4	5	4	3
2	-	-	-	5	5	6	5	4
3	-	-	-	-	4	5	4	4
4	-	-	-	-	-	5	4	3
5	-	-	-	-	-	-	5	4
6	-	-	-	-	-	-	-	4
7	-	-	-	-	-	-	-	-

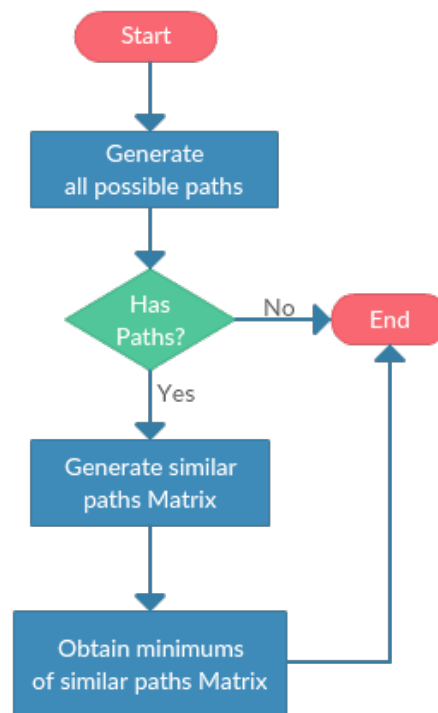


Figure 4.7: Best path algorithm flowchart based in common nodes minimization.

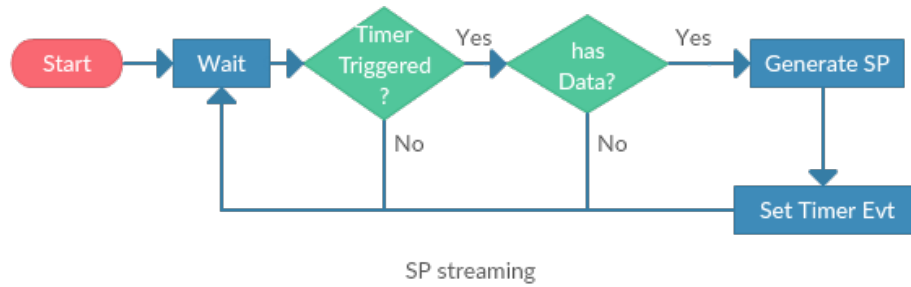


Figure 4.8: Streaming flowchart to generate Streaming packets.

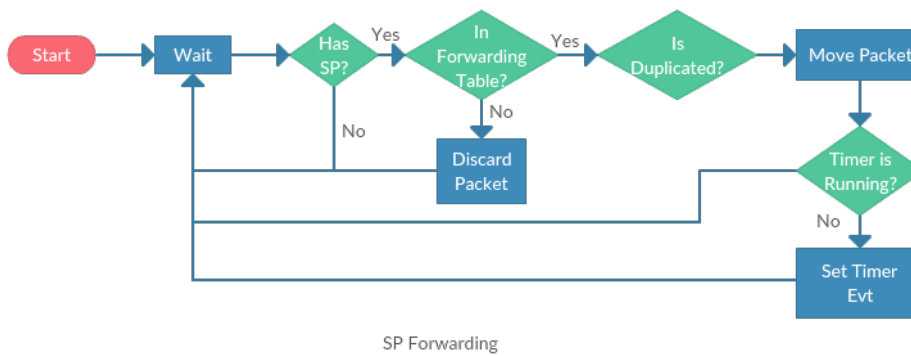


Figure 4.9: Forward flowchart to retransmit packets.

4.1.4.1 Streaming

During the Streaming state, the source node generates data to be send. The data handling for this process is represented in figure 4.8. The system waits for a send trigger and if there is data to be sent by the user, SP are put in the transmission waiting queue. After this the send trigger is reset in order to generate more SP later. The transmission waiting queue is handled in section 4.1.5.

4.1.4.2 Forward

Forward state as the function in retransmitting packets, and its flowchart is represented in figure 4.9. After a SP has been received, belongs to the node forwarding table and was not previously received, the packet is moved to a waiting transmission queue, explained in section 4.1.5. Once the packet is moved it is necessary to verify if the timer which provides send event is running or not. Because the timer is set by the first allowed packet it receives and takes into account the processing duration of the packet to be forwarded.

4.1.4.3 Store

The Store state flowchart is presented in figure 4.10, the system awaits for a packet reception event. If the sender SP are declared at the sink node forwarding table and the packet is not a duplicate of an already received ones, then data is stored by sequentially and handled to the consumer. The packets are discarded and the system awaits for more. The

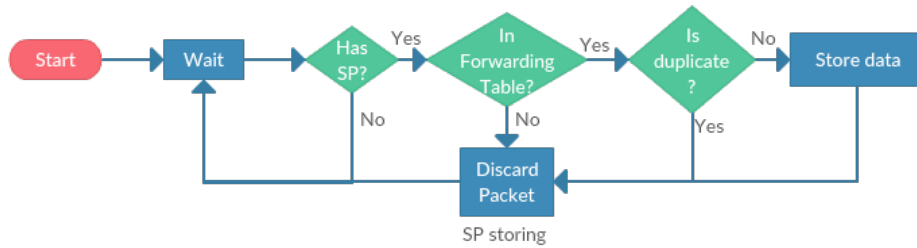


Figure 4.10: Store flowchart to provide the user with the received data stream.

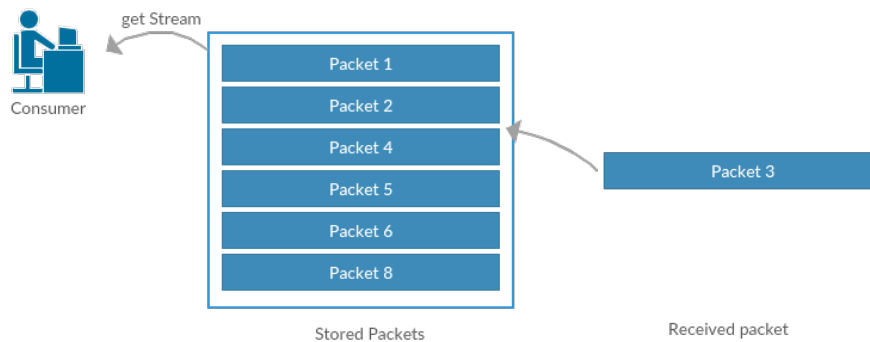


Figure 4.11: Sequential packet insertion.

user is only allowed to get packets only if there are fixed minimum packets stored, once to order the received packets it is a few are required in the buffer. This is shown in figure 4.11.

4.1.4.4 Time Division Multiplexing

To provide time division, the best paths must be known. After having this, slot allocation can take place to allocate slots to nodes. The slots duration should consider into account the transmission duration and a guard duration to overcome the non ideal characteristics of the platform in use. In the network example from figure 3.4 the allocation can take place by hop or path. Assuming the allocation algorithm taken into account was by path, the top time frame from figure 3.8 is generated. When node 3 receives its first packet from node 2, its transmission slot is supposed to start one slot time ahead, therefore it sets its transmission to one slot duration ahead, adjusted to the processing delays. Other example is node 4, when it receives a packet from node 1, it sets its transmission to 3 TDMA slots ahead from node 1 first transmission, once again adjusted from the packet processing delay.

4.1.5 Transmission and Reception

Transmission and Reception provide the reception and send operation events at the previous presented state machine.

The transmission is done every time the send event timer is triggered. This timer is set by the previously discussed flowcharts. When the timer is triggered, the node checks whether it has a packet or not by inspecting the transmission waiting queue. If true, the packet is sent,

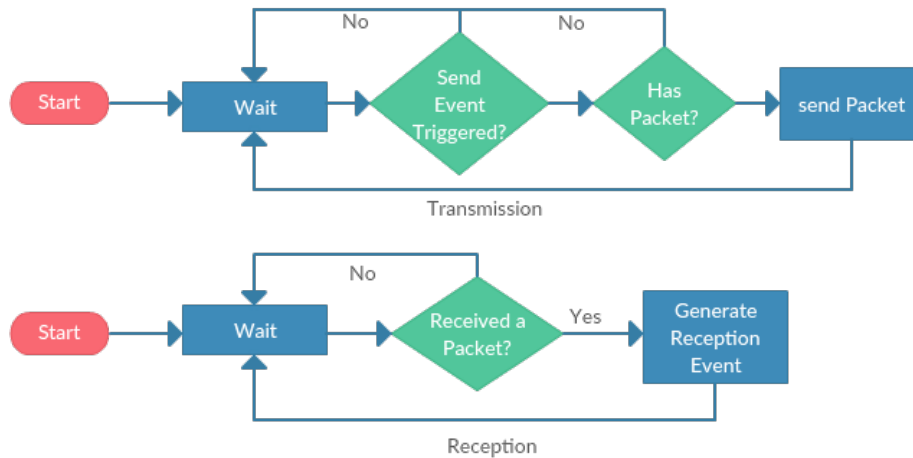


Figure 4.12: Transmission and Reception flowchart.

else returns to the waiting process until the timer is triggered once more.

In the reception, system waits for a packet to be received and when received, the system generates a reception event and at the same time, the packet is being put in a waiting queue to be processed.

4.2 Protocol Usage

This section aims to teach the WSRT protocol usage in a suitable operating system, by using the example provided in appendix A. The example can be executed by using the listing 4.1.

Listing 4.1: Protocol execution

```

1 Usage: $ sudo .\WsrtSoftware -i <iface> -id <id> -t <typ> [-e <err>] [-p <file>]
2
3 Eg: $ .\WsrtSoftware -i wlan0 -id 2 -t middle

```

The following options are passed to the software. It is mandatory to provide the wireless interface name, ID and node type.

- **-i** - to provide the wireless interface name which will be used.
- **-id** - to specify the ID given to the node.
- **-t** - to specify the node type. Which can be divided into three types: source, middle or storing node.
- **-e** - to specify the system error rate. It will make the system randomly discarding packets according to the assigned error.
- **-p** - file location containing user defined paths. The first line of this file must contain the amount of paths declared in it, while other lines are filled with defined paths. If this command is not used, paths will be dynamically generated.

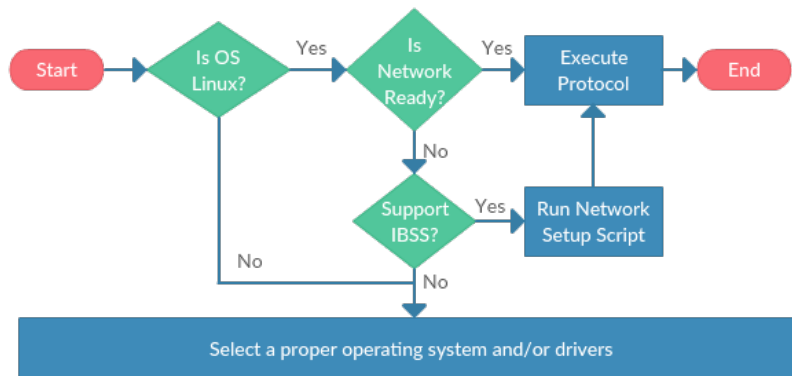


Figure 4.13: Necessary steps to implement the protocol.

Before the protocol execution, the OS must satisfy the requirements and be already pre-configured. This is described in section 4.2.1. To smooth the software distribution through all nodes in the network, debugging and monitoring tools are presented in section 4.2.2.

4.2.1 Requirements and Setup

To allow the user to use the WSRT protocol, the flowchart of figure 4.13 needs to be completed. If the OS is Linux, the system execute the protocol, assuming it supports Independent BSS (IBSS) mode and was previously pre-configured, otherwise the user should get a suitable operating system. A bash script was generated to setup the OS. The script main objective is to assign a static IP address to remotely control every single node, which will be discussed in section 4.2.2.1, and to create or join to an Ad-Hoc network at the OS boot. This process is explained in appendix B.

4.2.2 Tools

This section aims to describe the procedure by remotely controlling the system to transmit, compile and start 4.1 of the software to every single node in the network, described in section 4.2.2.1. A monitorization software was made for debugging purposes, called "Wsrst-Monitor" which will be discussed in section 4.2.2.2.

4.2.2.1 Remote Connection and File Transference

To be able to control each node remotely Cluster SSH should be used. This allows to control all the nodes OS as if they were a single node, by only inputing the commands into one terminal that will send them to every node. To execute Cluster SSH a file must be created having declared each node OS user credentials and ip. After this, the user needs to input the command presented in the listing 4.2 and is able to control all the nodes remotely. This procedure, as well as the cluster file name, is explained in appendix C.

Listing 4.2: Cluster SSH execution

```
1 $ cssh cluster
```

To upload an updated software version to every node, SSH File Transfer Protocol (SFTP) is used. To implement this a bash script is used. This procedure is described in D.

4.2.2.2 Network Monitorization

Network monitorization provides the means for capturing all WSRT protocol packets only.

When captured, the system records the time at which it has been received and puts it into a buffer. When the buffer has some amount of packets, these are transferred and stored in a text file allowing the user to debug the network if problems arise or to evaluate the protocol performance.

To use the network monitorization program the listing 4.3 in which it is necessary to pass as argument the wireless interface name which will monitor the network and also the file name to store the data.

Listing 4.3: Command to monitor the network

```
1 $ sudo .\WsrtMonitor -i <iface> -n <fileName>
```

With the obtained dataset some MATLAB scripts were created to process the file contents and generate human understandable data, which was used, for example, to gather the data used to perform evaluation, presented in chapter 5.

Results and Analysis

To evaluate the performance of the implemented spatial redundancy and time division multiplexing protocol, a few experiments were conducted. These experiments were conducted in a small laboratory environment, in which all nodes were connected. The test network is represented in figure 5.1.

At the source node, the protocol is executed in a laptop with Arch Linux and preemption enabled. Middle and Sink nodes are based on Raspberry Pi Zero, B, B+ and 3 in which Raspberry 3 has also preemption enabled. Finally, the monitor is a computer with Xubuntu 14.04 which captures and stores packets into a text file, for later processing with Matlab.

The laptops use its internal wireless interface, while the Raspberries use an USB wireless dongle. Nodes 2 and 3 wireless interface were cheap wireless dongles. Nodes 4, 5 and 6 use two TP-Link High Gain and Cisco wireless interface.

5.1 Network Initialization

In this section, Network Initialization, the system was initiated and executed until convergence and path distribution.

The experiment was broadcasting packets with 1500 bytes with connectivity matrices and forwarding tables with a transmission interval of 100 ms with a 10% variation around this value. Each state, Announcement and Path Definition, had the duration of 5 seconds to make sure the network had converged and paths were distributed.

In the Announcement state, at the convergence point all nodes were broadcasting the same connection matrix. The procedure to handle the results related to the Announcement state is provided in appendix E. The system was executed with different error rates, to simulate different operational conditions and 10 tests were made for each error rate. The errors were introduced at the reception layer of nodes, which discarded packets randomly, with a given probability.

The obtained results are shown in table 5.1 and table 5.2 as well figure 5.2. In figure 5.2 an increment of the convergence time with the error rate was expected to be seen, this did not happen, despite the fact that for each of the error rates 10 tests were made. The reason for this can be traced to the uncontrollable packet losses, since it is broadcasting into air, the medium can be perturbed. This hypothesis is confirmed when comparing table 5.1 with

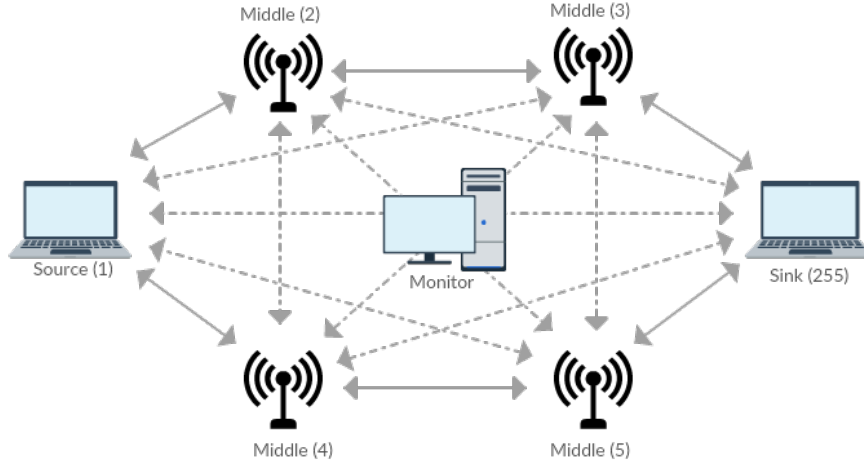


Figure 5.1: Network implemented in laboratory for experiments, the monitor node captures all the network traffic and is placed about the same distance to every node.

Table 5.1: Network convergence time.

Induced Error	0%	2%	4%	8%	12%	16%	20%
Node 1	539 ms	634 ms	614 ms	540 ms	553 ms	672 ms	1034 ms
Node 2	621 ms	679 ms	741 ms	648 ms	698 ms	730 ms	1109 ms
Node 3	637 ms	779 ms	696 ms	700 ms	643 ms	688 ms	1120 ms
Node 4	595 ms	656 ms	629 ms	606 ms	584 ms	685 ms	1053 ms
Node 5	580 ms	649 ms	664 ms	589 ms	559 ms	672 ms	1074 ms
Node 225	609 ms	713 ms	677 ms	626 ms	618 ms	687 ms	1074 ms
Maximum	637 ms	779 ms	741 ms	700 ms	698 ms	730 ms	1120 ms

table 5.2. The nodes which provide the maximum convergence time are directly associated with the higher packet losses. For example in the experiment with induced error of 4% the convergence time is around 779 ms and the correspondent node has a packet loss of 40%. Convergence time is flat between 5% and 15% because of the lack of tests.

During the Path Definition state, since all nodes were in direct contact with each other, the path distribution should not take more than the transmission interval imposed, around 100 ms. If bigger, it is due to packet losses.

Table 5.3 has relevant information about the path distribution duration. This table was generated with a MATLAB script described in appendix F. Node 1 is not included since it was the one which provided the best paths. In this table it is important to notice, that in every test the distribution time does not bring significant changes, due to every node being in direct contact. When source node, node 1, transmits the path and every other node receives it, the receiving node will transmit it once more and at this point the monitoring node captures the time when this happens.

Some distribution duration values are smaller than the minimum established transmission interval, less than 90 ms. This happens because the send event timer was already set at the Announcement state and the system had already transited to the Path Definition state in which it will resend a path packet right after it triggers.

Table 5.2: Announcing packets loss till convergence.

Induced Error	0%	2%	4%	8%	12%	16%	20%
Node 1	3.6%	3.2%	6.6 %	3.6%	1.75%	1.5%	18%
Node 2	29.8%	28.6%	33.3%	31.3%	39.2%	40.4%	28.2%
Node 3	29.1%	40.4%	37.3%	42.3%	31.9%	29.4%	38.5%
Node 4	2.2%	6.1%	8.5%	6.5%	6.8%	17.6%	9.5%
Node 5	6.5%	2%	15.3%	6.4%	4.4%	5.7%	8.9%
Node 225	6.6%	11.8%	16.3%	15.2%	27.5%	43.9%	2.7%

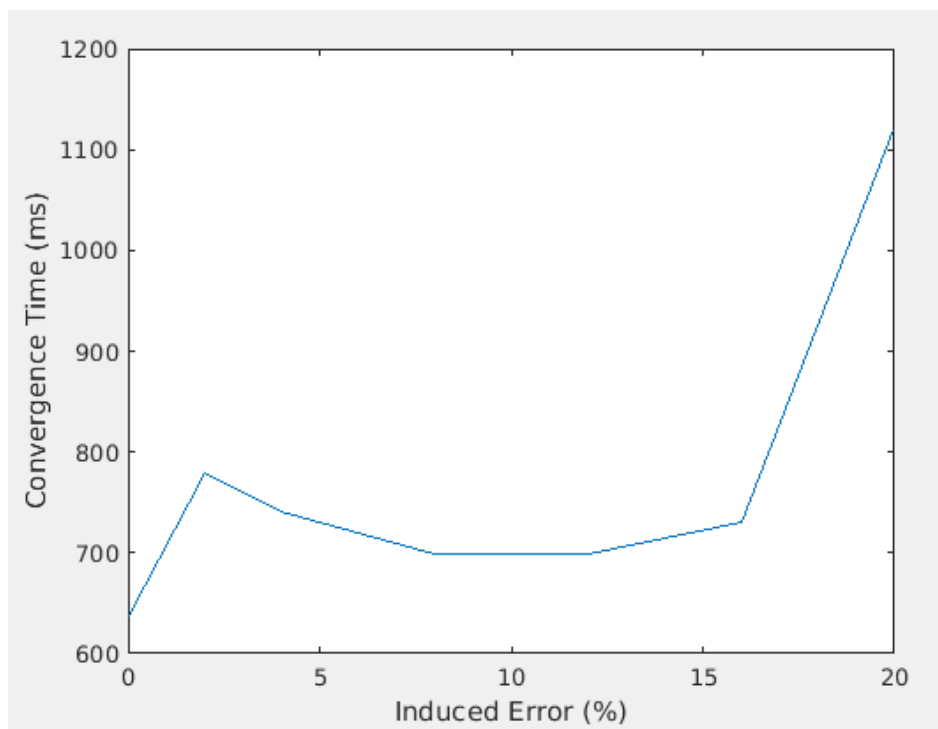


Figure 5.2: Announcing Convergence with relation to packet losses.

Table 5.3: Network path distribution duration.

Induced Error	0%	2%	4%	8%	12%	16%	20%
Node 2	126 ms	150 ms	179 ms	199 ms	175 ms	120 ms	209 ms
Node 3	155 ms	175 ms	176 ms	158 ms	330 ms	131 ms	224 ms
Node 4	133 ms	151 ms	145 ms	108 ms	126 ms	113 ms	142 ms
Node 5	90 ms	124 ms	142 ms	149 ms	78 ms	69 ms	107 ms
Node 6	77 ms	139 ms	175 ms	163 ms	114 ms	78 ms	110 ms
Maximum	155 ms	175 ms	176 ms	199 ms	330 ms	131 ms	224 ms

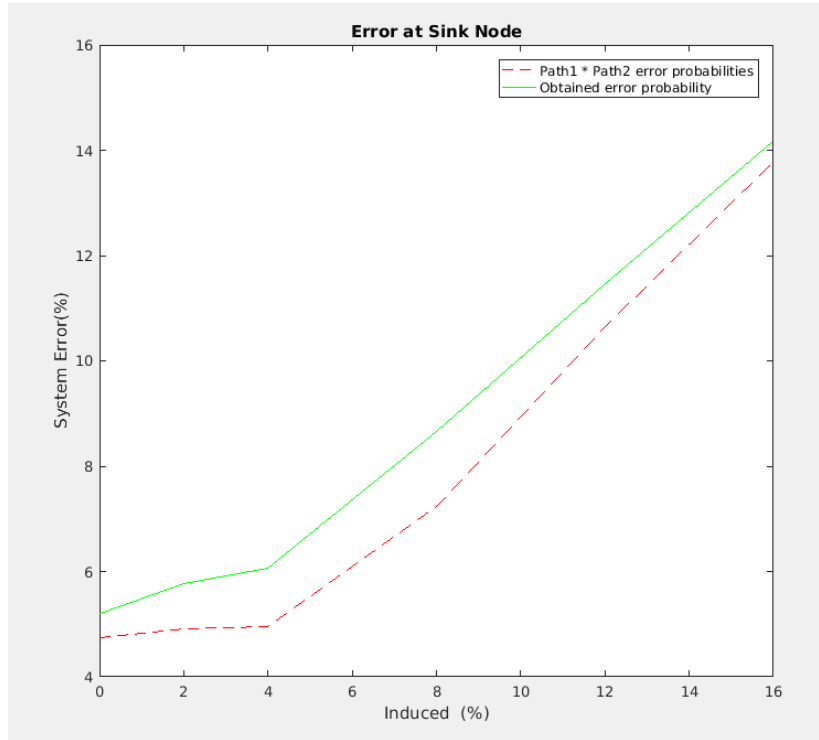


Figure 5.3: System Error with independent paths.

5.2 Running Mode

These are the most important test results, once they assess the overall performance of the protocol developed in the scope of this dissertation. It is said that the protocol, WSRT is aimed to provide spatial redundancy to increase the packet reception at the sink node and also time division to schedule transmissions. This section also evaluates retransmissions variation, paths jitter and system bandwidth.

The slot base time is 100 ms, and frame duration is based on the active transmitting nodes in the network, in this case, nodes 1,2,3,4,5, which generates a time frame of 500 ms. The results in this section were processed by a MATLAB script presented in appendix G.

5.2.1 Spatial Redundancy

The results in this section aim to verify spatial redundancy, being represented in Table 5.4. The table describes the path error and the overall system error when having 2 paths. With the induced error incrementation, the path error probabilities were expected to increment, and that does not happen between 0% and 4%.

Experiments were repeated several times, but the result was the same. This can be explained by the induced error rate being very low, to the dominant error source.

With the same data flowing through path 1 and path 2 to the sink node, probability of receiving the data increases. If the path probabilities were independent, which they are not, because they share the same environment and can influence it, by multiplying path 1 to path 2 errors probabilities, the result is near the system error. This comparison can be visualized in

Table 5.4: Table of packet loss percentage for each path and overall system error.

Induced Error	0%	2%	4%	8%	12%	16%
Path 1	22.5%	29.69%	32.02%	32.82%	37.12%	41.35%
Path 2	21.1%	16.51%	15.46%	22.02%	28.69%	33.29%
System	5.2%	5.77%	6.06%	8.65%	11.46%	14.16%

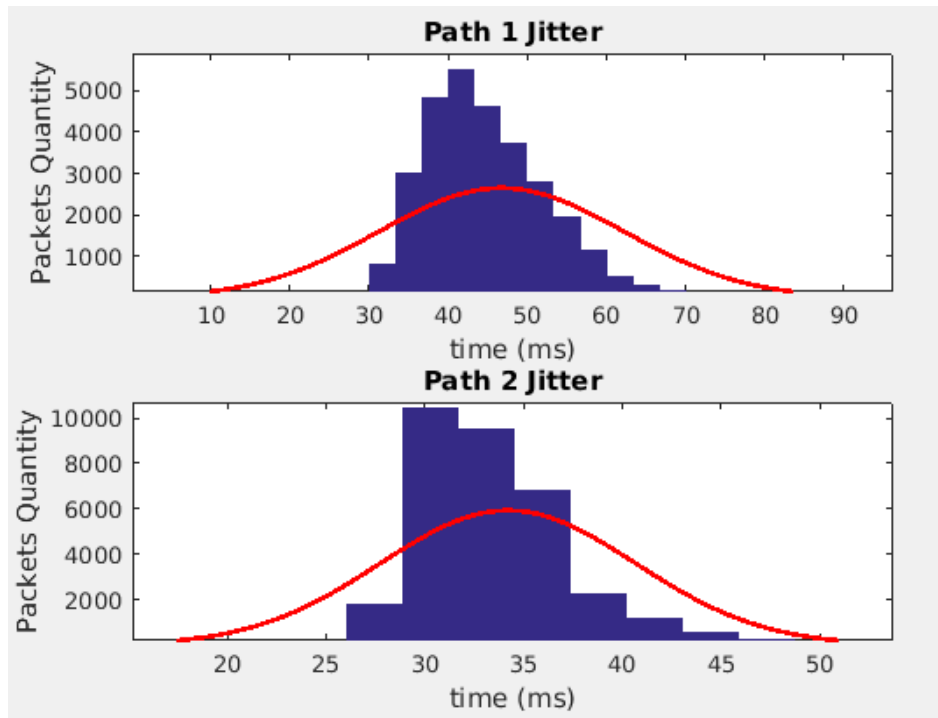


Figure 5.4: Paths jitter.

figure 5.3 which relates independent path probabilities error, with the measured probability.

5.2.2 Paths Jitter

During system normal operation mode, the nodes will not transmit exactly at the specified time instant calculated using the allowed packet reception. This is because of the uncontrollable background OS tasks and the platforms not being suitable for this kind of application. This is shown in figure 5.4 where the most probable jitter for path 1 is between 40 to 45 ms and path 2 between 30 to 35 ms. Path 1 was formed by Raspberry Pi Zero and B+, while path 2 was formed by a Raspberry Pi Zero and 3. Raspberry Pi 3 had a preemptive OS with four cores and so it caused less 10 ms of jitter. The maximum likely paths jitter can also establish a minimum slot duration.

If jitter was bigger than the slot duration, node 5, could still be transmitting at the same time node 1 is starting a new frame, resulting in a possible collision. This is verified in section 5.4.

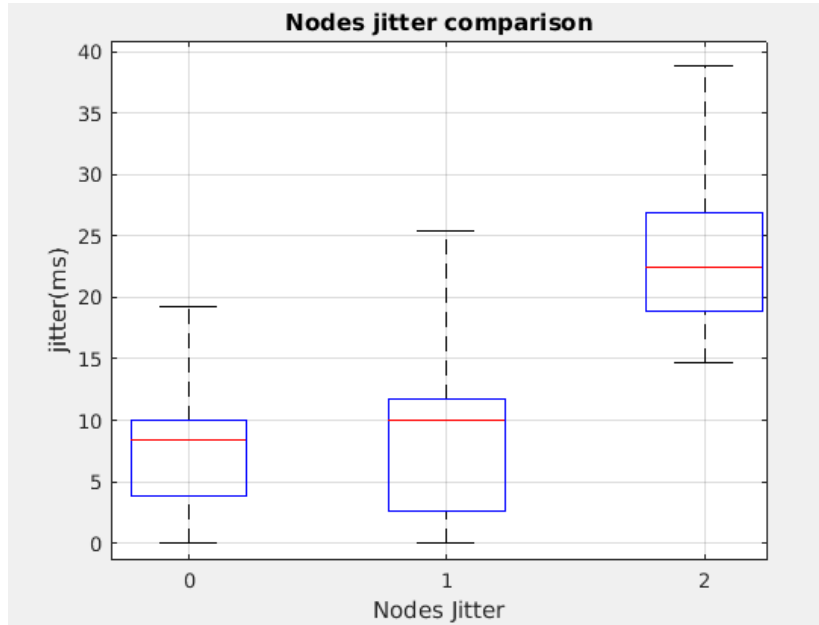


Figure 5.5: System performance regarding to the transmission between 2 neighbor nodes.

5.3 Jitter between consecutive nodes

In this tests, only 2 nodes were used, node 1 and 2, both running on Raspberry Pi, with the monitor node to track jitter between nodes but also the period deviations between transmissions. Slot duration for each node is set to 100 ms with only one packet transmitted each slot. The sampling lasted 70 minutes and the results are displayed in figure 5.5. The first and second box, from left to right, represents the maximum, average and minimum transmission jitter every time each node sends a packet, respectively. The last box, represents the jitter between node 2 sending a packet and node 3 retransmitting it. Node 2 has a average jitter of 7 ms, node 3 has 10 ms and between nodes is 23 ms. These results were processed with the MATLAB script presented in appendix H.

5.4 Maximum achievable data rate

For each test, the system was programmed to transmit a fixed amount of packets for each slot to obtain the maximum possible network bandwidth. In this case five tests were conducted for 1, 2, 3, 4 and 5 packet transmissions for each slot with 100 ms of slot duration.

During this tests processing, a delay was discovered for multiple packet transmissions at the same slot on every Raspberry Pi nodes, although this delay was not visible using a laptop which transmitted all the packets in less than one millisecond interval. The dataset for two packets per slot was used to obtain this detected delay. These results were processed with the MATLAB script presented in appendix I. This dataset generated the figure 5.6 in which the maximum, minimum and average delay are presented. The average delay between same slot transmissions is 14 ms. The main source of this delay could be hidden in the Raspberry Pi OS system tick which generates interruptions with every 10 ms. Other delay source could be the wireless dongle connected to the Raspberry Pi USB hub.

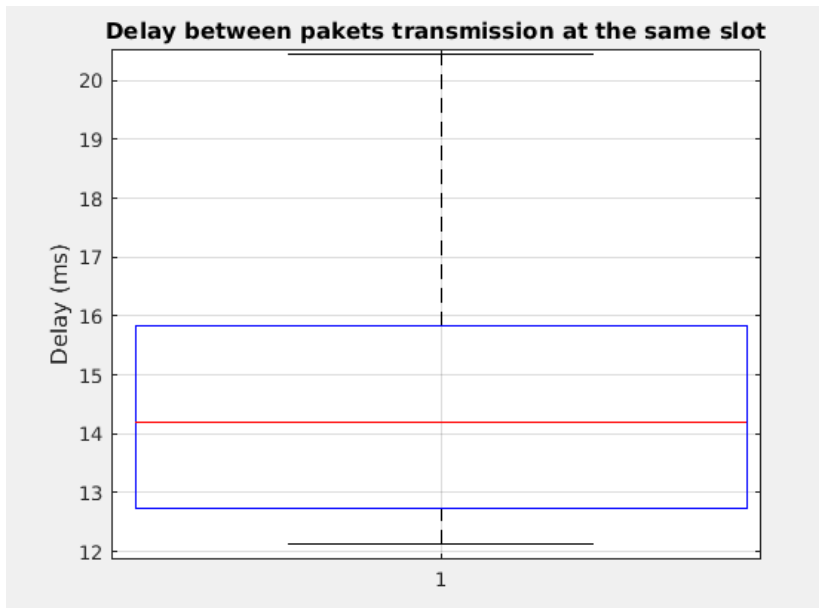


Figure 5.6: Delay during multiple packet transmissions in the same slot.

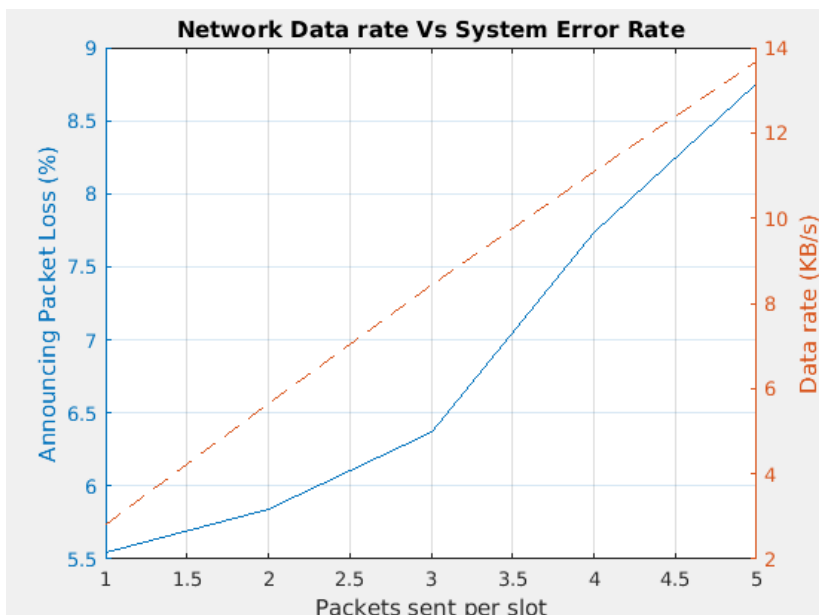


Figure 5.7: Network data rate and system error rate.

Table 5.5: Time Slots Overlapping

Time Stamp	Node ID	Sequence Number
17.78967	001	0a
17.80785	001	0b
17.82180	001	0c
17.83697	001	0d
17.85095	001	0e
17.88494	005	05
17.89742	005	06
17.91077	005	07
17.92326	005	08
17.93575	005	09
17.97229	002	0a
17.98476	002	0b
17.99881	002	0c
18.02660	002	0e
18.16375	003	0a
18.17629	004	0a
18.18882	003	0b
18.20260	004	0b
18.21504	003	0c
18.22994	004	0c
18.24240	004	0d
18.26986	003	0e
18.28233	004	0e

In figure 5.7, the straight line represents the obtained network data rate, while the dashed line represents the system error rate. The system error rate increasing is due to slot overlapping of consecutive TDMA frames presented in table 5.5. According to the TDMA frame, the transmissions are done sequentially following the transmission order of 1, 2, 3, 4, 5. It is possible to verify that node 5 is always transmitting packets from the previous frame due to node 1 and 5 slot overlapping. Due to this, node 1 and 5 packets collide and are lost.

The overlapping origin is caused by the path cumulative jitter for each transmission. For example, in path 1, 4 and 5, according to section 5.3, the average jitter between nodes is on average 23 ms. Transmission of multiple packets in the same slot generates a delay of 14 ms proportional to the packets quantity. So at node 5 transmission starts with an offset of 69 ms and when transmitting more than 3 packets this surpasses the defined slot period of 100 ms. Due to this, node 1 starts transmitting a new frame while node 5 is still transmitting. The error rate after 3 packets sent per slot starts to greatly increase due to overlapping, so the maximum data rate should be smaller than 8.5 KB/s, considering that the theoretical is 9 KB/s, once the used TDMA frame has the duration of 500 ms and each frame transmits 3 packets of 1500 bytes.

5.5 Results conclusions

According to the presented results, spatial redundancy and time division multiplexing mechanism improve the system performance having verified low error rates through multiple paths when compared with errors only using a single one. Although video and audio streaming cannot be achieved using the current platforms, since it is needed at least 300 Kbps for low resolution video streaming, as presented in table 2.2 and the system has a maximum data rate of 68 Kbps, or 8.5 KB/s.

Low data rates obtained are related to every cumulative jitter through the paths and also the delay when transmitting multiple packets in the same slot, caused by the Linux system tick, which cause the time division multiplexing mechanism to fail.

The increase the data rate obtained in this dissertation IEEE 802.11 frame structure which allows to transmit more data in the same frame could be applied to overcome the payload limit of 1500 bytes.

Another proposed solution is a usage of a more powerful platform, a laptop, once the source node, running in a laptop, was sending multiple packets at its own frame with small time impact over all the system, below millisecond delay.

Conclusion

The actual objective of this dissertation was to develop a system to transmit via wireless multimedia in lossy environments. To solve this a protocol was created using spatial redundancy in which each packet is forwarded through "n" disjunct paths, with "n" being defined by the user. It is used a TDMA mechanism to avoid collisions. The protocol was implemented in low cost hardware, in this case Raspberry Pi.

The experimental results verify the protocol satisfies the main objective, but the platform characteristics block the practical usage of the system, since it wastes too much bandwidth.

Possible project improvements are presented in section 6.1, which could solve the bandwidth efficiency and also insert new nodes and delete others.

6.1 Future Work

This project can still be greatly improved by applying some of the following modifications:

1. Protocol implementation at kernel level to reduce processing latency.
2. Implementation of IEEE 802.11 frame to increase the amount of data being sent and possibly increasing transmission rate.
3. Monitor tool, to have a real time visualization over the overall system performance and nodes.
4. Implementation of preemptive patch to the middle nodes operating system to have a more precise transmission.
5. The system should run a more complex algorithm to allow multiple nodes to the same slot if this nodes are not in range.
6. Dynamic node ID attribution based in MAC addresses, which requires a modification to the announcement state where MAC addresses are sent instead of node ID and between Announcement state and Path Definition, having a intermediary node to inform which ID was given to the node.

7. Automatic best path update, to allow the inclusion of new nodes and remove nodes that fail permanently.

Bibliography

- [1] James Kurose and Keith Ross. *Computer Networking - A top-down approach*. 2013.
- [2] IEEE Computer Society. What is 802.11 doing? <http://www.ieee802.org/11/presentation.html>, 2015.
- [3] Andrew S Tanenbaum. *Computer Networks*. 1996.
- [4] Youtube. Recommended upload encoding settings. <https://support.google.com/youtube/answer/1722171?hl=en>.
- [5] Behrouz A. Forouzan and Sophia Chung Fegan. *Data Communications and Networking*. 2007.
- [6] William Stallings. 2007.
- [7] Frederico Santos, Luís Almeida, and Luís Seabra Lopes. Self-configuration of an adaptive TDMA wireless communication protocol for teams of mobile robots. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pages 1197–1204, 2008.
- [8] Zigbee Alliance. What is ZigBee. <http://www.zigbee.org/what-is-zigbee/>. [Online; accessed 15-05-2017].
- [9] D. Chen et al. *WirelessHART: Real-Time Mesh Network for Industrial Automation*. Springer, 2010.
- [10] Bluetooth. Discover Bluetooth. <https://www.bluetooth.com/what-is-bluetooth-technology/discover-bluetooth>. [Online; accessed 15-05-2017].
- [11] IEEE Computer Society. Ieee standard for ethernet. <http://standards.ieee.org/getieee802/download/802.3-2015.zip>, 1995.
- [12] Microsoft. What Is 802.11 Wireless? [https://technet.microsoft.com/en-us/library/cc757419\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc757419(v=ws.10).aspx). [Online; accessed 23-06-2017].
- [13] Shahin Farahani. *ZigBee Wireless Networks and Transceivers*. 2008.
- [14] Digi. Digi XBee® ZigBee.

-
- [15] FieldCommGroup. HART - DIGITAL TRANSFORMATION FOR ANALOG INSTRUMENTS. <https://fieldcommgroup.org/technologies/hart>. [Online; accessed 15-05-2017].
- [16] Jianping Song, Song Han, Aloysius K. Mok, Deji Chen, Mike Lucas, Mark Nixon, and Wally Pratt. WirelessHART: Applying wireless technology in real-time industrial process control. *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 377–386, 2008.
- [17] Raspberry Pi. Products. <https://www.raspberrypi.org/products/>. [Online; accessed 15-05-2017].
- [18] Hardkernel. Products. <http://www.hardkernel.com/main/main.php>. [Online; accessed 15-05-2017].
- [19] Kernel. About Linux Kernel. <https://www.kernel.org/linux.html>. [Online; accessed 15-05-2017].
- [20] Libby Clark. Intro to Real-Time Linux for Embedded Developers. <https://www.linux.com/blog/intro-real-time-linux-embedded-developers>, 2013. [Online; accessed 04-07-2017].
- [21] Xenomai. How does Xenomai deliver real-time? <https://xenomai.org/start-here/>. [Online; accessed 15-05-2017].
- [22] Dave Marshall. Video and audio compression. <https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node200.html>, 1995.
- [23] Andy Beach. *Video Compression*.
- [24] Toni Fiedler. The aac audio coding family for broadcast and cable tv. https://www.iis.fraunhofer.de/content/dam/iis/de/doc/ame/wp/FraunhoferIIS_White-Paper_AAC-Broadcast-CableTV.pdf.

Protocol Usage Example

To use this protocol the user should use the example code provided in listing A.1 to stream data.

Listing A.1: WSRT usage example.

```
1 #include <stdio.h>
2 #include <string.h>
3 #include "Wsrt/Wsrt.h"
4
5 int main(int argc, char **argv){
6     int i;
7     char *interface;;
8     uint8_t type = 0;
9     uint8_t id = 0;
10    uint8_t mode = PATHS_DYNAMIC;
11    uint8_t err = 0;
12    char *fileName;
13    setbuf(stdout, NULL);
14
15    // Read arguments
16    for(i = 0; i < (argc); ++i){
17        if(strcmp(argv[i], "-i")==0){
18            interface = malloc(sizeof(argv[i+1]));
19            interface = argv[i+1];
20            printf("[Testing] Interface: %s\n", argv[i+1]);
21        }
22        if(strcmp(argv[i], "-t")==0){
23            if(strcmp(argv[i+1], "source") == 0){
24                printf("[Testing] Source Node inserted.\n");
25                type = WSRT_SOURCE;
26            }
27            if(strcmp(argv[i+1], "sink") == 0){
```

```

28     printf("[Testing] Sink Node inserted.\n");
29     type = WSRT_SINK;
30 }
31 if(strcmp(argv[i+1],"middle") == 0){
32     printf("[Testing] Middle Node inserted.\n");
33     type = WSRT_MIDDLE;
34 }
35 }
36 if(strcmp(argv[i],"-path")==0){
37     fileName = argv[i+1];
38     mode = PATHS_STATIC;
39 }
40 if(strcmp(argv[i],"-id")==0){
41     id = atoi(argv[i+1]);
42 }
43 if(strcmp(argv[i],"-e")==0){
44     err = atoi(argv[i+1]);
45     printf("ERROR: %d",err);
46 }
47 }
48 printf("[Testing] initializing Wsrt.\n");
49 Wsrt *wsrtPtr = wsrt_init(interface ,id ,type ,PACKET_SIZE);
50
51 if(type == WSRT_SOURCE && mode == PATHS_STATIC){ // if it is source
52     node and mode is declared as dynamic, default is dynamic
53     printf("[Testing] Mode set to Static - %s\n",fileName);
54     wsrt_setPathMode(wsrtPtr ,mode, fileName);
55 }
56 printf("[Testing] initializing threads.\n");
57 wsrt_initThreads(wsrtPtr);
58 wsrt_setError(wsrtPtr ,err);
59
60 uint32_t counter = 0;
61 uint8_t dataStream[wsrtPtr->packetLength-WSRT_DATA_POS];
62 uint8_t buff[PAYLOAD_SIZE];
63 printf("Waiting for network to be ready.\n");
64
65 while(wsrt_ready(wsrtPtr) == 0);
66 printf("Network is ready...\n");
67
68 while(1){
69     if(type == WSRT_SOURCE){ // Generate data every 100 ms
70         memcpy(dataStream,&counter ,sizeof(uint32_t));
71         memset(dataStream + sizeof(uint32_t) ,0 ,wsrtPtr->packetLength-
72             WSRT_DATA_POS-sizeof(uint32_t));
73         counter++;

```

```
72     wsrt_write(wsrtPtr, dataStream, wsrtPtr->packetLength-
       WSRT_DATA_POS);
73     usleep(100000);
74 }
75 else if(type == WSRT_SINK){ // Get data whenever there is a
       packet.
76     while(wsrt_hasNext(wsrtPtr)){
77         if(wsrt_read(wsrtPtr, buff))
78             perror("No payload to return.\n");
79         uint32_t count;
80         memcpy(&count, buff, sizeof(uint32_t));
81         printf("SEQ %d\n", count);
82     }
83 }
84 }
85 return 0;
86 }
```

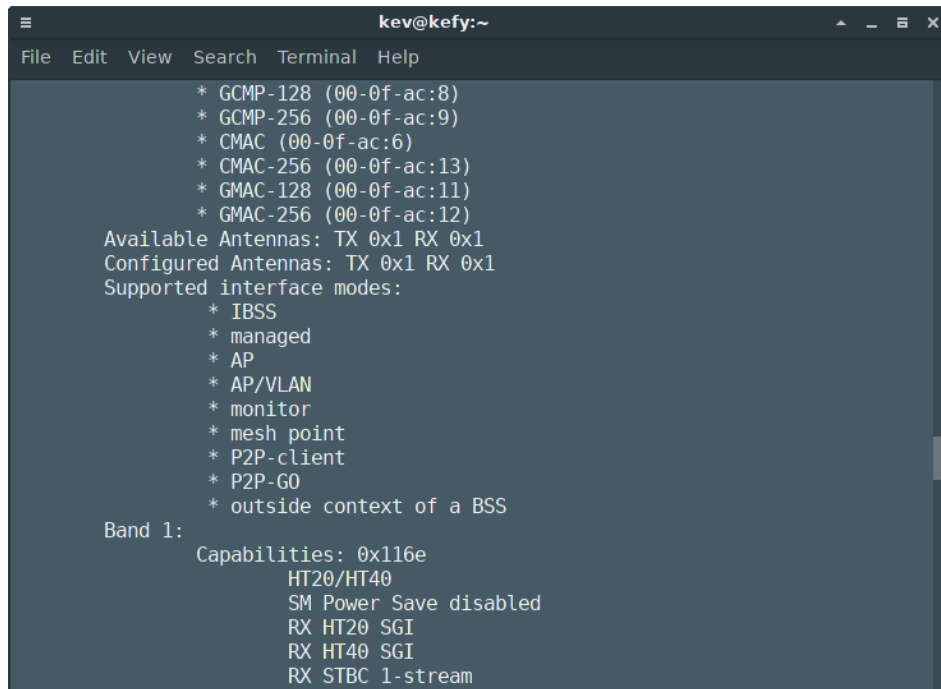

Operating System Configuration Instructions

This appendix aims to provide the operating system necessary configuration to handle the created protocol. It is expected the user already has an Arch Linux operating system installed in the device which will use. This instructions were made to be done in a Raspberry Pi.

First it is necessary to check if USB dongle which provides wireless communication supports an interface mode called IBSS. This is done by using in the Linux terminal the following command:

```
1 $ iw list
```

This command provides the output presented in figure B.1, in which the supports IBSS interface mode. If this doesn't appear, means that the used interface is not capable of handling the protocol.



```

kevf@kefy:~
File Edit View Search Terminal Help
* GCMP-128 (00-0f-ac:8)
* GCMP-256 (00-0f-ac:9)
* CMAC (00-0f-ac:6)
* CMAC-256 (00-0f-ac:13)
* GMAC-128 (00-0f-ac:11)
* GMAC-256 (00-0f-ac:12)
Available Antennas: TX 0x1 RX 0x1
Configured Antennas: TX 0x1 RX 0x1
Supported interface modes:
* IBSS
* managed
* AP
* AP/VLAN
* monitor
* mesh point
* P2P-client
* P2P-GO
* outside context of a BSS
Band 1:
Capabilities: 0x116e
HT20/HT40
SM Power Save disabled
RX HT20 SGI
RX HT40 SGI
RX STBC 1-stream

```

Figure B.1: Check for IBSS mode support.

To configure the OS, a configuration script was made. This script provides the automatic startup of a mesh network. This script is called `WsrtConfig` and it is made by other small scripts, in which one of them generates an ad-hoc start script which will be run every time the OS system boots up.

Listing B.1: Preparation script for OS.

```

1
2 #!/bin/bash
3 ##### ./WsrtConfig.sh file #####
4 echo -e "———— Install Openssh ————"
5 #yes | pacman -Syu openssh
6 echo -e "———— Add sshd service for ssh ————"
7 systemctl enable sshd.service
8 systemctl start sshd.service
9
10 echo -e "———— Using ./addHosts.sh $(cat hostArgs) ————"
11 ./addHosts.sh $(cat hostArgs)
12 echo -e "———— Using ./generateAdHocScript.sh $(cat adhocArgs)————
   "
13 ./generateAdHocScript.sh $(cat adhocArgs)
14
15 cp adhoctStart.timer /etc/systemd/system/
16 cp adhoctStart.service /etc/systemd/system/
17
18 systemctl enable adhoctStart.timer

```

```
19 systemctl start adhoctStart.timer
```

Listing B.2: Change hosts accordingly to specified Node ID

```
1 #!/bin/bash
2 #### ./addHosts.sh file ####
3 while getopts p:h:d: option
4 do
5 case "${option}"
6 in
7 p) IP=${OPTARG};;
8 h) HNAME=${OPTARG};;
9 d) dm=${OPTARG};;
10 esac
11 done
12
13 host_entry=" $IP           $HNAME.$dm           $HNAME"
14 rm /etc/hostname
15 echo "$HNAME" >> /etc/hostname
16
17 display_usage() {
18 echo -e "Usage example:\n  $./addHosts.sh -p 127.0.1.1
19 -h NodeX -d localDomain"
20 }
21
22 echo "$host_entry"
23 if [[ -z $IP ]] || [[ -z $HNAME ]] || [[ -z $dm ]]
24 then
25 display_usage
26 exit 1
27 fi
28
29 if [ ! -z "$matches_in_hosts" ]
30 then
31 echo "Updating existing hosts entry."
32 # iterate over the line numbers on which matches were found
33 while read -r line_number; do
34 # replace the text of each line with the desired host entry
35 sed -i '' "${line_number}s/./${host_entry} /" /etc/hosts
36 done <<< "$matches_in_hosts"
37 else
38 echo "Adding new hosts entry."
39 echo "$host_entry" | tee -a /etc/hosts > /dev/null
40 fi
```

Listing B.3: Generates a ad-hoc script to be initialized on the OS system startup

```
1 #!/bin/bash
```

```
2 ##### ./generateAdHocScript.sh file #####
3 while getopts i:n:c:p:m:a option
4 do
5 case "${option}"
6 in
7 i) INTERFACE=${OPTARG};;
8 n) NETWORKNAME=${OPTARG};;
9 c) CHANNEL=${OPTARG};;
10 p) IP=${OPTARG};;
11 m) MASK=${OPTARG};;
12 esac
13 done
14
15 display_usage() {
16 echo -e "Usage example:\n  ./generateAdHocScript
17 -i wlan0 -n MeshPi -c 2412 -p 192.168.1.1 -m 24"
18 }
19
20 echo -e "#!/bin/bash\n
21 ip addr flush $INTERFACE\n
22 ip link set ${INTERFACE} down\n
23 iw $INTERFACE set type ibss\n
24 ip link set $INTERFACE up\n
25 iw $INTERFACE ibss join $NETWORKNAME $CHANNEL \n
26 ip addr add $IP/$MASK dev $INTERFACE" > /usr/local/bin/adhoc.sh
27 chmod 777 /usr/local/bin/adhoc.sh
```

Cluster SSH Instructions

To be able to use Cluster SSH it is necessary to have a configuration file located at "`\etc`" with the name of clusters. This file must contain every node information as presented in listing C.1

Listing C.1: Clusters config file

```
1 clusters wsrt@192.168.1.2 wsrt@192.168.1.3 wsrt@192.168.1.4  
wsrt@192.168.1.5 wsrt@192.168.1.6
```


Software Upload Instructions

To upload the software into every node in network containing 6 nodes the listing D.1 is used. This uploads the software using SFTP by using the SFTP commands declared in listing D.2, in which will put all the contents from the "BinSFTP" folder in every single node and exit. This folder must be in the same location as the "FirmwareUpload.sh" file.

Listing D.1: Firmware upload script

```
1 #!/bin/bash
2 ##### ./FirmwareUplod.sh file ###
3 echo "Node 1"
4 sshpass -p q sftp -oBatchMode=no -b binBatch wsrt@192.168.1.1
5 echo "Node 2"
6 sshpass -p q sftp -oBatchMode=no -b binBatch wsrt@192.168.1.2
7 echo "Node 3"
8 sshpass -p q sftp -oBatchMode=no -b binBatch wsrt@192.168.1.3
9 echo "Node 4"
10 sshpass -p q sftp -oBatchMode=no -b binBatch wsrt@192.168.1.4
11 echo "Node 5"
12 sshpass -p q sftp -oBatchMode=no -b binBatch wsrt@192.168.1.5
13 echo "Node 6"
14 sshpass -p q sftp -oBatchMode=no -b binBatch wsrt@192.168.1.6
```

Listing D.2: Preparation script for OS.

```
1 ##### binBatch file ###
2 put -r BinSFTP
3 exit
```


MATLAB script for announcing phase results processing

The following script was used to provide the time when each node converged. The overall system convergence was calculated by using the maximum of the nodes convergence. This script is used to process text files generated by the WsrtMonitor.

```

1 % Read txt file
2 tt = readtable(fileName, 'ReadVariableNames', true, 'Delimiter', ',', ',');
3
4 % Filter by debug message type
5 toDelete = tt.MsgType == 2;
6 fTT = tt(toDelete, :);
7 fTT(:, 2) = [];
8
9 % Read Packets Message Content
10 info = cell(length(fTT.Message), length(strsplit(char(fTT.Message(1)
    ))));
11 for i = 1 : length(fTT.Message)
12 info(i, :) = strsplit(char(fTT.Message(i)));
13 end
14 fTT.MacAddr = info(:, 1);
15 fTT.type = str2double(info(:, 2));
16 fTT.seq = str2double(info(:, 3));
17 fTT.src = hex2dec(info(:, 4));
18 fTT.Payload = reshape(hex2dec(info(:, 2:end)), [size(info, 1) size(
    info, 2) - 1]);
19 fTT.Message = [];
20
21 % Obtain Announcement Phase Data
22 toDelete = fTT.type == 1;
23 announcTable = fTT(toDelete, :);

```

```

24 announcTable.Payload(:,[1:3]) = [];
25
26 % Get Announcement Convergence and Loss
27 announcTable.plLength = sum((announcTable.Payload ~= 0),2); % for
    announcement
28 maxLength = max(announcTable.plLength); % Gives when convergence
    happens.
29 nodeList = unique(announcTable.src); % All nodes which
    broadcasted ID and connections tables
30 for i = 1 : length(nodeList)
31 node(i).id = nodeList(i);
32 toDelete = announcTable.src == nodeList(i);
33 node(i).announcPhase = announcTable(toDelete,:);
34 pos = find(node(i).announcPhase.plLength == maxLength);
35 convTime(i) = round((node(i).announcPhase.TimeStamp(pos(1)) - node(
    find(nodeList == 1)).announcPhase.TimeStamp(1))*1000);
36 maxPcktSeq = node(i).announcPhase.seq(pos(1));
37 pcktReceived = size(node(i).announcPhase(1:(find(node(i).
    announcPhase.seq == maxPcktSeq)),:),1);
38 txRxConv(1,i) = maxPcktSeq + 1; % Transmitted packets by node
39 txRxConv(2,i) = pcktReceived; % Received packets by the monitor
40 node(i).convTime = convTime(i);
41 end

```

MATLAB script for path definition phase results processing

The following script was used to provide the time when every node has the paths which data will be forwarded. The overall system distribution time was calculated by using the maximum of nodes time instant when it happened.

```

1 % Read txt file
2 tt = readtable(fileName, 'ReadVariableNames', true, 'Delimiter', ',', ',');
3
4 % Filter by debug message type
5 toDelete = tt.MsgType == 2;
6 fTT = tt(toDelete, :);
7 fTT(:, 2) = [];
8
9 % Read Packets Message Content
10 info = cell(length(fTT.Message), length(strsplit(char(fTT.Message
    (1)))));
11 for i = 1 : length(fTT.Message)
12     info(i, :) = strsplit(char(fTT.Message(i)));
13 end
14 fTT.MacAddr = info(:, 1);
15 fTT.type = str2double(info(:, 2));
16 fTT.seq = str2double(info(:, 3));
17 fTT.src = hex2dec(info(:, 4));
18 fTT.Payload = reshape(hex2dec(info(:, 2:end)), [size(info, 1) size(
    info, 2) - 1]);
19 fTT.Message = [];
20
21 % Divide Data by Path Definition Phase
22 toDelete = fTT.type == 2;
23 pathTable = fTT(toDelete, :);

```

```
24 pathTable.Payload(:,[1:3]) = [];  
25  
26 % Get Path Distribution and Loss  
27 nodeList = unique(pathTable.src);  
28 for i = 1 : length(nodeList)  
29     node(i).id = nodeList(i);  
30     toDelete = pathTable.src == nodeList(i);  
31     node(i).pathPhase = pathTable(toDelete,:);  
32 end  
33  
34 for i = 1 : length(nodeList)  
35     dTime(i) = node(i).pathPhase.TimeStamp(1);  
36     dTime(i) = round((dTime(i) - node(find(nodeList == 1)).pathPhase  
37         .TimeStamp(1))*1000);  
37     node(i).dTime = dTime(i);  
38 end
```

MATLAB script for running mode data processing

The presented script allows to verify the spatial redundancy and jitter induced by each path.

```

1  clear
2  %% Packets Processing
3  % Control variables
4  MAX_PCK_PROC = 50;
5  %% Read txt file
6  fileName = 'runTest_16_1.txt';
7  tt = readtable(fileName, 'ReadVariableNames', true, 'Delimiter', ',', ' ');
8  %% Filter by debug message type
9  toDelete = tt.MsgType == 2;
10 fTT = tt(toDelete, :);
11 fTT(:, 2) = [];
12 %% Read Packets Message Content
13 info = cell(length(fTT.Message), length(strsplit(char(fTT.Message
    (1)))));
14 for i = 1 : length(fTT.Message)
15     info(i, :) = strsplit(char(fTT.Message(i)));
16 end
17 fTT.MacAddr = info(:, 1);
18 fTT.type = str2double(info(:, 2));
19 fTT.seq = str2double(info(:, 3));
20 fTT.src = hex2dec(info(:, 4));
21 fTT.Payload = reshape(hex2dec(info(:, 2:end)), [size(info, 1) size(
    info, 2) - 1]);
22 fTT.Message = [];
23

```

```

24 %% Divide Data by network phases – Announcement, Path Definition,
    Running
25 toDelete = fTT.type == 3;
26 runTable = fTT(toDelete,:);
27 runTable.Payload(:,[1:3]) = [];
28
29 %% Running Data Processing
30 % Objectives:
31 %             -> Get Packet Loss for each path
32 %             -> Calculate Jitter of each path
33 %
34 runTable.streamNumb = 256.*runTable.Payload(:,2) + ...
35 runTable.Payload(:,1); % for announcement
36 nodeList = unique(runTable.src);
37 % organize tables to correspondent nodes.
38 for i = 1 : length(nodeList)
39     node(i).id = nodeList(i);
40     toDelete = runTable.src == nodeList(i);
41     node(i).runPhase = runTable(toDelete,:);
42 end
43
44
45 %% Calculate Packet Loss Per direct neighbours and per Path
46 path = [ 1 2 3 ;
47         1 4 5 ];
48 % Due to monitor computer reception errors, lets check how
49 % many packets the monitor received before the end node, which
50 % the monitor didn't receive from 1. those are packets
51 % which the network didn't lots. This is used to compensate
52 % the monitor packets losses. Because since a node
53 % forwarded a packet which wasn't detected by the node
54 % which transmitted.
55
56 maxPacketsSent = 0:1:max(node(1).runPhase.streamNumb);
57 monitorLoss = (1 - size(node(find(nodeList == 1)).runPhase,1) / ...
58 length(maxPacketsSent)) * 100;
59 for y = 1 : size(path,1)
60     for x = 2 : size(path,2)
61         node(find(path(y,x) == nodeList)).pktLoss = 100 * (1 - size(
62             node(find(path(y,x) == nodeList)).runPhase,1) / size(node(find
63                 (nodeList == path(y,x-1)).runPhase,1));
64         node(find(path(y,x) == nodeList)).lossLink = [path(y,x-1), path
65             (y,x)];
66     end
67 end
68
69 for i = 1 : size(node,2)

```

```

67     nodesList(i) = node(i).id;
68 end
69
70 for y = 1 : size(path,1)
71     node(find(255 == nodesList)).pktLoss(y,:) = 100 * (1 - ((size(
        node(find(path(y,end) == nodesList)).runPhase,1))/length(
        maxPacketsSent)));
72     node(find(255 == nodesList)).lossLink(y,:) = [path(y,1), path(y,
        end)];
73 end
74
75 %% Calculate overall packet loss by proving redundancy
76 SOURCE = 1;
77 if(node(1).id == SOURCE)
78     maxPckt = max(node(1).runPhase.streamNumb);
79 end
80
81 checkRpt = [];
82 for i = 1 : size(path,1)
83     checkRpt = [checkRpt ; node(find(nodesList == path(i,end))).
84     runPhase.streamNumb];
85 end
86
87 [~, ind] = unique(checkRpt);
88 duplicate_ind = setdiff(1:size(checkRpt, 1), ind);
89 totalDiffReceived = length(checkRpt) - length(checkRpt(
        duplicate_ind));
90
91 total = maxPckt;
92 error = 100 - (totalDiffReceived/total) * 100;
93
94 % Calculate Paths jitter
95 TDMA_baseTime = 0.1; % in seconds
96 TDMA_frame = [ 1 2 3 4 5 ];
97 TDMA_ofss = [0 0.1 0.1 0.3 0.1];
98 path = [ 1 2 3;
99         1 4 5];
100
101 for i = 1 : size(path,1)
102     checkRpt = [node(find(nodesList == path(i,1))).runPhase ; node(
        find(nodesList == path(i,end))).runPhase];
103     checkRpt = sortrows(checkRpt);
104     cnt = 0;
105     pathDiff = path(i,end) - path(i,1);
106     for x = 2 : length(checkRpt.TimeStamp)
107         if(checkRpt.streamNumb(x-1) == checkRpt.streamNumb(x))

```

```
108     pathDur(i) = (checkRpt.TimeStamp(x) - checkRpt.TimeStamp(x-1)
109                 );
109     cnt = cnt + 1;
110     pJitter(i,cnt) = pathDur(i) - pathDiff * TDMA_baseTime;
111     end
112 end
113 pJitter(i,1:end-1) = pJitter(i,2:end);
114 end
115 pJitter = pJitter';
```

Inter node jitter calculation

This appendix provides a MATLAB script to calculate the deviation after a node receives a packet and starts transmitting it.

```

1 %% Read txt file
2 tt = readtable('N1_N2_Jitter_1.txt', 'ReadVariableNames', true, '
    Delimiter', ',', ');
3 %% Filter by debug message type
4 toDelete = tt.MsgType == 2;
5 fTT = tt(toDelete, :);
6 fTT(:, 2) = [];
7
8 % Choose Running Mode Messages
9 info = cell(length(fTT.Message), length(strsplit(char(fTT.Message
    (1)))));
10 for i = 1 : length(fTT.Message)
11     info(i, :) = strsplit(char(fTT.Message(i)));
12 end
13 fTT.MacAddr = info(:, 1);
14 fTT.type = str2double(info(:, 2));
15 fTT.seq = str2double(info(:, 3));
16 fTT.src = hex2dec(info(:, 4));
17 fTT.Payload = reshape(hex2dec(info(:, 2:end)), [size(info, 1) size(
    info, 2) - 1]);
18 fTT.Message = [];
19
20 toDelete = fTT.type == 3;
21 runTable = fTT(toDelete, :);
22 runTable.Payload(:, [1:3]) = [];
23
24 % Divide transmissions by nodes
25 runTable.streamNumb = 256.*runTable.Payload(:, 2) + runTable.
    Payload(:, 1); % for announcement

```

```

26 nodesList = unique(runTable.src);
27 for i = 1 : length(nodesList)
28     node(i).id = nodesList(i);
29     toDelete = runTable.src == nodesList(i);
30     node(i).runPhase = runTable(toDelete,:);
31 end
32
33 % Calculate Jitter between 2 nodes
34 slotTime = 100; % 100 ms of slot time
35 path = [ 1 2 ];
36
37 % InterNode Jitter
38 pJitter = [];
39 for i = 1 : size(path,1)
40     checkRpt = [node(find(nodesList == path(i,1))).runPhase ; node(
41         find(nodesList == path(i,end))).runPhase];
42     checkRpt = sortrows(checkRpt);
43     cnt = 0;
44     pathDiff = path(i,end) - path(i,1);
45     for x = 2 : length(checkRpt.TimeStamp)
46         if (checkRpt.streamNumb(x-1) == checkRpt.streamNumb(x))
47             pathDur(i) = (checkRpt.TimeStamp(x) - checkRpt.TimeStamp(x-1)
48                 );
49             cnt = cnt + 1;
50             pJitter(i,cnt) = 1000*pathDur(i) - pathDiff * slotTime;
51         end
52     end
53     pJitter(i,1:end-1) = pJitter(i,2:end);
54 end
55 pJitter = pJitter';
56
57 % Imprecisions when in 1st node transmissions
58 % For Node 1
59 checkRpt = node(1).runPhase;
60 checkRpt = sortrows(checkRpt);
61 cnt = 0;
62 timeFrame = 500;
63
64 intervVarN1 = [];
65 for x = 2 : length(checkRpt.TimeStamp)
66     if ((checkRpt.streamNumb(x-1)+1) == checkRpt.streamNumb(x))
67         pathDur(i) = abs((checkRpt.TimeStamp(x) - checkRpt.TimeStamp(x
68             -1)));
69         cnt = cnt + 1;
70         intervVarN1(i,cnt) = abs(1000*pathDur(i)-timeFrame);
71     end
72 end

```

```

70  intervVarN1(i,1:end-1) = intervVarN1(i,2:end);
71  intervVarN1 = intervVarN1';
72
73  % For Node 2
74  checkRpt = node(2).runPhase;
75  checkRpt = sortrows(checkRpt);
76  cnt = 0;
77  timeFrame = 500;
78
79  intervVarN2 = [];
80  for x = 2 : length(checkRpt.TimeStamp)
81      if(((checkRpt.streamNumb(x-1)+1) == checkRpt.streamNumb(x)))
82          pathDur(i) = (checkRpt.TimeStamp(x) - checkRpt.TimeStamp(x-1)
83              );
84          if(abs(1000*pathDur(i)-timeFrame) < 300)
85              cnt = cnt + 1;
86              intervVarN2(i,cnt) = abs(1000*pathDur(i)-timeFrame);
87          end
88      end
89  intervVarN2(i,1:end-1) = intervVarN2(i,2:end);
90  intervVarN2 = intervVarN2';
91
92  jitt(1) = mean(pJitter);
93  jitt(2) = mean(intervVarN1);
94  jitt(3) = mean(intervVarN2);
95  vJit = [ intervVarN1 ; intervVarN2 ; pJitter ]
96  grp = [ zeros(1,length(intervVarN1)), ones(1,length(intervVarN2)
97          ), 2.*ones(1,length(pJitter)) ]';
98
99  h = boxplot(vJit,grp)
100  xlabel('Nodes Jitter');
101  ylabel('jitter (ms)');
102  title('Nodes jitter comparison')
103  set(h(7,:), 'Visible', 'off')
104  grid on

```




Bandwidth calculation

This script is used to evaluate the system maximum data rate and also deviations imposed when transmitting multiple packets.

```
1  t = readtable(fileName, 'ReadVariableNames', true, 'Delimiter', ',', ',');
2  %% Filter by debug message type
3  toDelete = tt.MsgType == 2;
4  fTT = tt(toDelete, :);
5  fTT(:, 2) = [];
6  %% Read Packets Message Content
7  info = cell(length(fTT.Message), length(strsplit(char(fTT.Message
      (1)))));
8  for i = 1 : length(fTT.Message)
9      info(i, :) = strsplit(char(fTT.Message(i)));
10     end
11     fTT.MacAddr = info(:, 1);
12     fTT.type = str2double(info(:, 2));
13     fTT.seq = str2double(info(:, 3));
14     fTT.src = hex2dec(info(:, 4));
15     fTT.Payload = reshape(hex2dec(info(:, 2:end)), [size(info, 1) size(
      info, 2) - 1]);
16     fTT.Message = [];
17
18     % Divide Data by network phase - Running
19     toDelete = fTT.type == 3;
20     runTable = fTT(toDelete, :);
21     runTable.Payload(:, [1:3]) = [];
22
23     runTable.streamNumb = 256.*runTable.Payload(:, 2) + runTable.
      Payload(:, 1); % for announcement
24     nodesList = unique(runTable.src);
25     % organize tables to correspondent nodes.
26     for i = 1 : length(nodesList)
```

```

27     node(i).id = nodesList(i);
28     toDelete = runTable.src == nodesList(i);
29     node(i).runPhase = runTable(toDelete,:);
30     end
31
32
33     % Obtain start timeStamp
34     startTm = node(1).runPhase.TimeStamp(1);
35     lastTm = node(5).runPhase.TimeStamp(end);
36     dur = lastTm - startTm;
37
38     % Packets sent
39     maxPckt = max(node(1).runPhase.streamNumb);
40
41     % Obtain system received packets
42     checkRpt = [];
43     pcktSize = 1500;
44     overHead = 17;
45     checkRpt = [node(5).runPhase.streamNumb ; node(3).runPhase.
46         streamNumb];
47     [~, ind] = unique(checkRpt);
48
49         % indices to unique
50         values in column 3
51     duplicate_ind = setdiff(1:size(checkRpt, 1), ind);
52         % duplicate indices
53     receivPckts = length(checkRpt) - length(checkRpt(duplicate_ind));
54         % succesfull packets received without duplicates
55     data(1) = (1 - receivPckts/maxPckt) * 100;
56     data(2) = (receivPckts * pcktSize)/(dur * 1000);
57         % rate in kbps
58     data(3) = (receivPckts * (pcktSize-overHead))/(dur * 1000);
59
60 %% Inter packets Jitter.
61 tt = readtable(fileName, 'ReadVariableNames', true, 'Delimiter', ',', ' ');
62 % Filter by debug message type
63 toDelete = tt.MsgType == 2;
64 fTT = tt(toDelete,:);
65 fTT(:,2) = [];
66 % Read Packets Message Content
67 info = cell(length(fTT.Message), length(strsplit(char(fTT.Message
68     (1)))));
69 for i = 1 : length(fTT.Message)
70     info(i,:) = strsplit(char(fTT.Message(i)));
71 end
72 fTT.MacAddr = info(:,1);
73 fTT.type = str2double(info(:,2));
74 fTT.seq = str2double(info(:,3));

```

```

67 fTT.src = hex2dec(info(:,4));
68 fTT.Payload = reshape(hex2dec(info(:,2:end)),[size(info,1) size(
    info,2)-1]);
69 fTT.Message = [];
70
71 % Divide Data by network phase – Running
72 toDelete = fTT.type == 3;
73 runTable = fTT(toDelete,:);
74 runTable.Payload(:,[1:3]) = [];
75
76 % remove source from data processing because it is a laptop
77 toDelete = fTT.src == 1;
78 runTable = fTT(toDelete,:);
79 runTable.Payload(:,[1:3]) = [];
80
81 runTable.streamNumb = 256.*runTable.Payload(:,2) + runTable.
    Payload(:,1); % for announcement
82 cnt = 0;
83
84 for x = 1 : length(runTable.streamNumb) - 1
85     if((runTable.streamNumb(x)+1 == (runTable.streamNumb(x + 1))) &&
        (runTable.src(x) == runTable.src(x+1)))
86         differe(x,:) = [((runTable.TimeStamp(x+1) - runTable.TimeStamp(
            x))*1000) runTable.src(x) runTable.streamNumb(x) runTable.src
            (x+1) runTable.streamNumb(x + 1)];
87         if(differe(x,1) < 50)
88             cnt = cnt + 1;
89             interPcktJitter(cnt) = differe(x,1);
90         end
91     end
92 end
93
94 h = boxplot(interPcktJitter)
95 ylabel('Delay (ms)');
96 title('Delay between pakets transmission at the same slot')
97 set(h(7,:), 'Visible', 'off')
98
99 grid on

```

