SCALABLE AND INTERFERENCE AWARE WI-FI MESH NETWORKS USING COTS DEVICES

Pedro Miguel Salazar Teixeira Júlio

Departamento de Engenharia Eletrotécnica

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Telecomunicações

2015

This report fulfills the needed requirements of the course Tese/Dissertação (Thesis) from the 2nd year of the Master in Electronics and Computer Engineering.

Student: Pedro Miguel Salazar Teixeira Júlio, Nr. 1080521, 1080521@isep.ipp.pt
Scientific Orientation: Prof. Dr. Jorge Botelho da Costa Mamede, jbm@isep.ipp.pt
Company: Intituto de Engenharia de Sistemas e Computadores – Tecnologia e Ciência (INESC TEC)
Supervision: Prof. Dr. Rui Lopes Campos, rcampos@inesctec.pt
Co-supervision: Eng.º Filipe Ribeiro, filipe.a.ribeiro@inesctec.pt

Electrical Engineering Department
Master in Electronic and Computer Engineering
Specialization in Telecommunications
2015

To my family and girlfriend

Acknowledgements

A lot of the success of this thesis comes from hard-work, effort and perseverance, but were undoubtedly the people that surrounded all the way who had the hugest impact on the work here presented.

First, I would like to express my sincere gratitude to my supervisor, Prof. Dr. Rui Campos, for the vote of confidence, shared knowledge and for allowing me to join such an interesting and challenging project, and be a part of such an impressive working group at INESC TEC.

To my advisor, Prof. Dr. Jorge Mamede another huge appreciation for his readiness to accept to be my advisor. Also all his shared knowledge, tips and support that guide me through the development of this thesis.

Another big appreciation to Filipe Ribeiro for all the valuable help, support, guidance and who was ever available to remove any doubt at any time. Also for all the other members of Wireless Netwoks (WiN) group, Filipe Teixeira and Mário Lopes for their readiness and valuable tips and José Oliveira and Sérgio Conceição for sustaining the productive environment.

Also my thanks to Jouni Malinen developer of *hostapd* for quick answering my emails and tips.

To my girlfriend that unconditionally supported me throughout all these years and who has the never left my side for second and always gave me the strength needed pursue my goals and do better.

To my family, for all the support, and sustain they provided me, allowing me to pursue my goals and fully focus on my work.

Also to my friends and colleagues for all support and leisure time that allowed to recover and decompress from the hard work

Pedro Júlio

Resumo

A crescente tendencia no acesso móvel tem sido potenciada pela tecnologia IEEE 802.11. Contudo, estas redes têm alcance rádio limitado. Para a extensão da sua cobertura é possível recorrer a redes emalhadas sem fios baseadas na tecnologia IEEE 802.11, com vantagem do ponto de vista do custo e da flexibilidade de instalação, face a soluções cabladas.

Redes emalhadas sem fios constituídas por nós com apenas uma interface têm escalabilidade reduzida. A principal razão dessa limitação deve-se ao uso do mecanismo de acesso ao meio partilhado *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA) em topologias multi-hop. Especificamente, o CSMA/CA não evita o problema do nó escondido levando ao aumento do número de colisões e correspondente degradação de desempenho com impacto direto no throughput e na latência.

Com a redução da tecnologia rádio torna-se viável a utilização de múltiplos rádios por nó, sem com isso aumentar significativamente o custo da solução final de comunicações. A utilização de mais do que um rádio por nó de comuniações permite superar os problemas de desempenho inerentes ás redes formadas por nós com apenas um rádio.

O objetivo desta tese, passa por desenvolver uma nova solução para redes emalhadas multi-cana, *duar-radio*, utilizando para isso novos mecanismos que complementam os mecanismos definidos no IEEE 802.11 para o estabelecimento de um *Basic Service Set* (BSS). A solução é baseada na solução WiFIX, um protocolo de routing para redes emalhadas de interface única e reutiliza os mecanismos já implementados nas redes IEEE 802.11 para difundir métricas que permitam à rede escalar de forma eficaz minimizando o impacto na performance. A rede multi-hop é formada por nós equipados com duas interfaces, organizados numa topologia hierárquica sobre múltiplas relações *Access Point* (AP) – *Station* (STA).

Os resultados experimentais obtidos mostram a eficácia e o bom desempenho da solução proposta face à solução WiFIX original.

Palavras-Chave: Wi-Fi, Redes emalhadas, JCAR, Dual-radio, Beacon, Escalabilidade,

Abstract

The increasing trend on mobile access has been mainly potentied for IEEE 802.11 technology. However these networks suffer from reduced radio range. The extension of coverage can be potentiated by mesh deployments since they provide an ease, robust, flexible and cost effective solution for this problem. These networks are built upon nodes scattered in a mesh topology that form the backbone of an extended basic service set.

Single radio Wireless Mesh Networks (WMN) however suffer from reduced scalability. The main reason to such limitation is the use of Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) in the multi-hop topology. Specifically, CSMA/CA fails to prevent the hidden and exposed node occurrence, which respectively, lead to an increase on the number of collisions and flow retentions. The direct impact on throughput and latency reduces the overall network performance to values that no longer match user increasing demands.

As radio technology becomes cheaper, it became possible to equip nodes with multiple interfaces and operate them in multiple channels in order the reduce interference from links operating on a common channel.

Therefore the goal of this thesis is to develop a new WMN Multi-Radio Multi-Channel (MRMC) solution addressing new mechanisms not yet covered in state of art. The proposed solution, is based on WiFIX, a Single Radio (SR) WMN routing protocol and reuses the mechanisms already implemented in IEEE 802.11 networks to broadcast metrics that enable the network to auto-configure efficiently and to scale with minimum overhead. The multi-hop backbone is formed by nodes equipped with two interfaces disposed in a hierarchical topology, under multiple Access Point (AP) - Station (STA) relations.

The results obtained from an experimental testbed clearly show the effectiveness of the solution compared with the original WiFIX and its capability to scale resulting from the overhead control and co-channel interference reduction.

Key-Words.: Wi-Fi, Wireless Mesh Network, JCAR, Dual-radio, Beacons, Scalability

Contents

List of Figures

List of Tables

List of Code excerpts

Abbreviations

ACS	–	Automatic Channel Selection
ATCM	–	Active Topology Creation and Maintenance
AODV	–	Ad-hoc On-demand Distance Vector
API	–	Application Programming Interface
BATMAN	–	Better Approach To Mobile Ad-hoc Networking
BSS	–	Basic Service Set
BSSID	–	Basic Service Set Identifier
DCF	–	Distributed Coordination Function
CH	–	Cluster Head
CMS	–	Channel Management Server
COTS	–	Comodity of the Shelf
DMA	–	Direct Memory Access
DRU	–	Designated Radio Unit
DSR	–	Dynamic Source Routing
EBSS	–	Extended Basic Service Set
Eo11	–	Ethernet over 802.11
ETT	–	Expected Transission Time
G-PaMeLa	–	Generalized Partitioned Mesh Network Traffic and Interfere Aware Channel Assignment

HCH – Head of Cluster Head

IBSS – Independent Basic Service Set

ICMP – Internet Control Message Protocol

IE – Information Element

IEEE – Institute of Electrical and Electronics Engineers

ILP – Integer and Linear Programming

IP – Internet Protocol

IPC – Inter-Process Communication

JMR – Joint Multi-Rate

JRCA – Joint Routing Channel Assignment

LLP – Link Layer Protocol

LRA – Link-Rate Allocation

MAC – Medium Access Control

MCCA – Maxflow–based Centralized Channel Assignment

MEATT – Multi-channel Expected Anypath Transmission Time

MInLP – Mixed Integer Linear Programming

MIPS – Microprocessor without Interlocked Pipeline Stages

MR – Multi-Radio

MRMC – Multi-Radio Multi-Channel

NIC – Network Interface Card

OGM – Originator Messages

OR – Opportunistic Routing

OUI – Organizationally Unique Identifier

QoE – Quality of Experience

RCART – Robust joint Channel Assignment and Routing with Time Partitioning

RREP – Route Reply

RREQ – Route Request

RSSI – Received Signal Strength Indicator

RU – Radio Unit

SBC – Single Board Computer

SC – Single Channel

SDK – Software Development Kit

SINR – Signal to Interference Ratio

SR – Single Radio

SWR – Split Wireless Router

TDMA – Time Domain Multiple Access

TIC – Topology and Interference-aware Channel selection

TIM – Traffic Indication Map

UCI – Unified Configuration Interface

UWB – Ultra Wide Band

WCETT – Weighted Comulative Expected Transmission Time

WME – Wireless Multimedia Extensions

WMN – Wireless Mesh Networks

1. INTRODUCTION

1.1. SCOPE

Wireless technology is currently a fast growing market driven by the demand of mobile Internet access. Institute of Electrical and Electronics Engineers (IEEE) 802.11 based networks are now ubiquitous providing not only Internet access to mobile users, but also last mile network services in areas where cable technology is hardly an option, be it rural or disaster areas or temporary infrastructures. In any of these scenarios the network needs to be scalable and flexible in order to be able to recover from failures and fairly and efficiently transport traffic. Wireless Mesh Networks (WMNs) are usually employed since they are a cost-effective solution with all the capabilities stated.

A WMN is a static multi-hop wireless network composed of multiple Mesh Access Points (MAPs) placed between mesh clients and the wired infrastructure. It consists of a static Ad-hoc network where most of the traffic is directed to and from the wired infrastructure. Each MAP operates both as transmitter, receiver and relay forwarding packets to/from clients from/to other MAPs in the network. The main function of these devices is to provide Internet access to mesh clients. It is also possible to support other type of networks through a mesh gateway, as for example sensor networks, cellular, IEEE 802.11, IEEE 802.15 (WPAN / Bluetooth), IEEE 802.16 (WiMAX). Mesh connectivity is dynamically maintained throughout the network operation due to the capacity of self-configuration and self-healing

around broken or congested links. An example of the architecture of a WMN is presented in Figure 1.

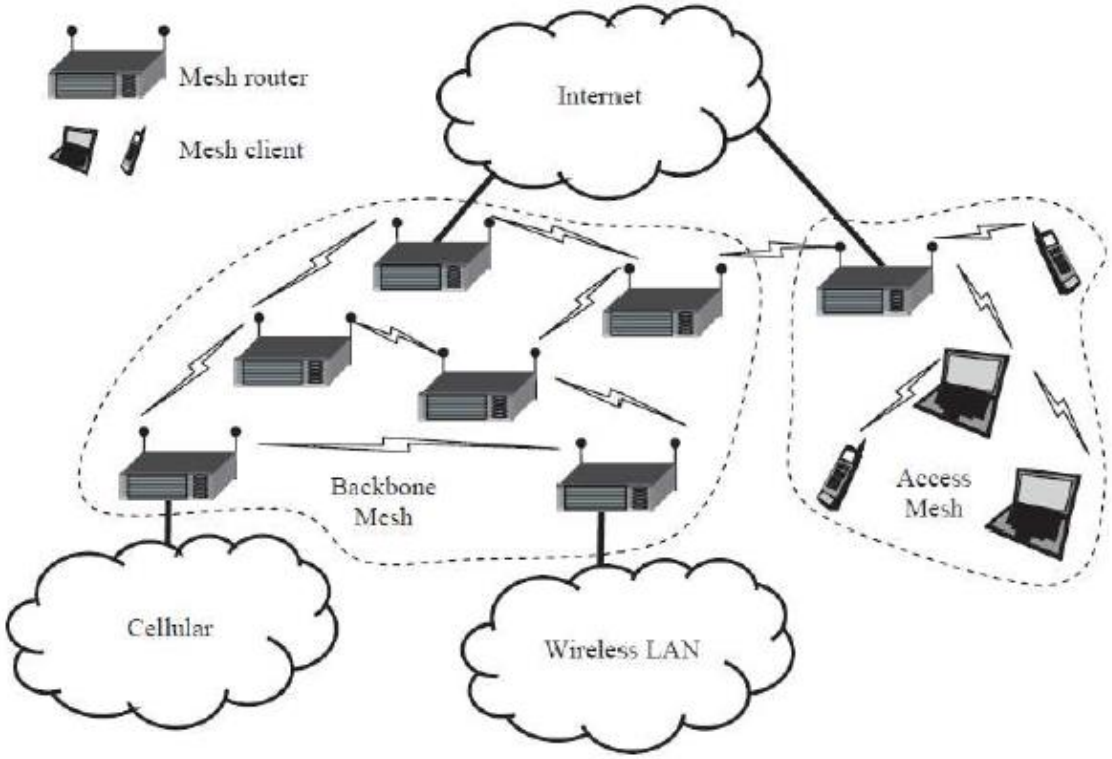


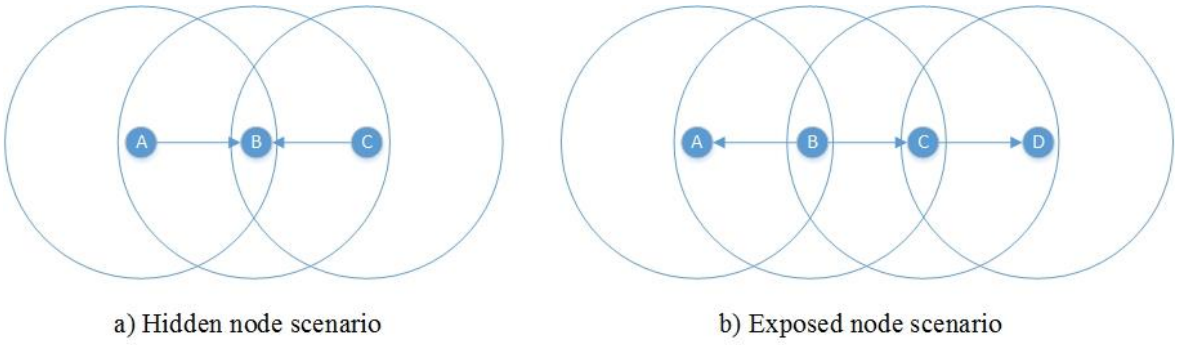
Figure 1 – Example of mesh architecture [1]

1.2. PROBLEM STATEMENT AND MOTIVATION

As the user demand increases so does the bandwidth requirement and the need to achieve high Quality of Experience (QoE). Wireless deployments often rely on commodity hardware to support the infrastructure. However due to the medium access mechanism, the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), these networks suffer from reduced scalability in multi-hop scenarios. CSMA is the mechanism implemented in wireless networks with the purpose of preventing frame collisions, where each node senses its environment before transmitting. Nonetheless, CSMA fails to prevent packet collision due to the hidden node problem in such topologies, which is exacerbated as the network grows. It occurs when terminals A and C are hidden from each other, in other words, the action range of both stations don't allow them to sense the other's transmission. One such scenario is depicted in Figure 2 a). Node A is transmitting its data to B. Node C, also in action range

of B and with data to send, senses the environment and by detecting that it's idle transmits, causing a collision at B which will not be detected by node A.

The IEEE 802.11 Distributed Coordination Function (DCF), based on CSMA also defines the Request-to-Send/Clear-to-Send (RTS/CTS) transmission scheme to prevent the hidden node problem. RTS/CTS handshake, provides time slot reservation to stations with high priority data to transmit. Upon listening to an RTS or CTS message, all the other terminals within range of both transmitter and receiver will be aware of the channel occupancy and hold back their transmissions until the recipient issues the acknowledgment of the data. In spite of the improvement to the hidden node problem, it fails to prevent the exposed node problem, which in fact is worsened. The exposed node problem, depicted in Figure 2 b) occurs when a station is prevented from transmitting due to a neighboring active link where the receiver is out of reach. In such scenario a parallel transmission could occur without interfering with the one already established, although because a station senses the medium busy it holds back the packets. The same would happen if a station hears a CTS from a neighbor, which would put it into a backoff period during which no data would be exchanged or RTS answered even if issued from stations to which no interference results.



a) Hidden node scenario b) Exposed node scenario

Figure 2 – Hidden and exposed node problems

Generally, both of these problems will lead to a reduction on the overall efficiency, augmenting the number of retransmissions that result in a lower data rate and high latency. With increasing number of active devices and increasing user demands the number of collisions also increases and this will cause a significant drop of network capacity and scalability. The direct impact on multi-hop networks performance, such as mesh or ad-hoc, is obvious if we consider the number of simultaneous active links and network density. Furthermore, RTS/CTS handshake has an underlying assumption that all hidden nodes are

in transmission range of receivers, that is true for most infrastructure deployments, but generally fails in the multi-hop scenario [2].

Single-Radio (SR) WMN often operate over a Single Channel (SC). Thus given the broadcast nature of the wireless medium a forwarding node will interfere with the subsequent nodes. The interference range in these networks is considered much larger than the communication range, as represented by (1) [3]:

$$Interference\ Range = 2 * Communication\ Range \quad (1)$$

An example of interference experienced in multi-hop networks is shown in Figure 3, where nodes 1, 2 4 and 5, that belong to the same path as 3 are within its interference range and nodes 8, 9 and 10 from a neighboring path also fall within interference range of 3. As a result, none of these nodes will be able to transmit simultaneously with node 3.

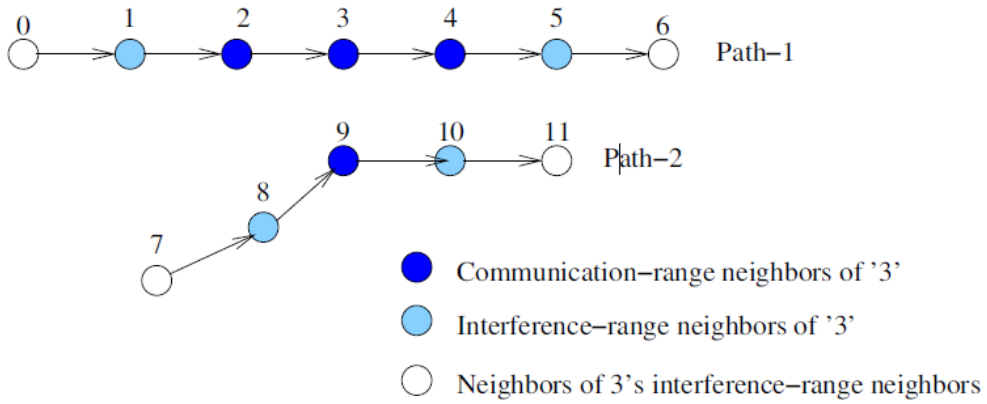


Figure 3 – Intra and Inter-path interference in single channel multi-hop networks [3]

In order to work around the interference limitation, multiple channels can be used. IEEE 802.11b/g and IEEE 802.11a standards, provide a total of 32 channels (according to 5GHz band as regulated by ETSI 301 893 [4]) on Industrial Scientific and Medical (ISM) 2.4GHz and 5GHz bands, among which 3 and 19 channels of each respective band are non-overlapping. Non-overlapping channels, permit the establishment of parallel transmissions without interference. However the channel switching times and the need to update channels for each sender-receiver pair in these round-robin schemes make these networks also not so efficient. A simple work around these problems is to equip nodes with multiple network interface cards (NICs).

Multi-Radio Multi-Channel (MRMC) WMNs are a cost-effective solution since radio technology has become cheaper. Equipping each node with multiple radios and operating them on multiple channels reduces interference and latency resulting in a much more scalable network with increased throughput. Although the multi-radio operation is not straight-forward and multiple factors should be taken in to account, mostly concerning routing and Channel Assignment (CA). Routing and CA are not independent problems. Solving the routing problem requires prior knowledge of the available bandwidth on all links. However, the available bandwidth is determined by channel assignment since the raw bandwidth of a channel is divided by the number of links sharing that same channel. Likewise, a CA algorithm needs to be aware of the expected load on each link, so that the bandwidth assigned to each link matches or outperforms the required flow rate.

1.3. OBJECTIVES

The main objective of this thesis is to study the state of art concerning mesh networks and perform a critical analysis to the current proposed solutions. Specifically, we will address methodology, strengths and weaknesses of each, that will lead us to the second objective, the outline of a preliminary solution capable of scale and fill a gap in the current state of art. Finally, we will develop and evaluate a MRMC WMN Joint Channel Assignment and Routing (JCAR) solution that extending the WiFIX routing scheme to the MRMC scenario allows the network to efficiently scale while reducing excessive control overhead.

1.4. DISSERTATION OUTLINE

In the current chapter is given the context, challenges, motivation and the main objectives of this thesis.

Chapter 2 describes the state of art concerning MRMC mesh schemes. This chapter is further divided into schemes that perform channel assignment, routing and other schemes that jointly address both of these challenges. At the end of each sub-chapter a brief discussion on the topic is provided.

Chapter 3 describes the IEEE 802.11 wireless protocol and its architecture under Linux Operating System.

The proposed solution is introduced in Chapter 4 were we describe the methodology used to insert metrics into beacons and disseminate control information with minimal overhead, so as process of the network self-configuration including the channel assignment and routing process.

The assembled testbed, used to evaluate the proposed solution's performance is presented in Chapter 5 along with results obtained concerning a set of metrics

Finally, on Chapter 6, the conclusion of the carried work during this dissertation are addressed along with future work

2. STATE OF ART

Within the scope of this work, it was researched the state of art solutions concerning MRMC WMNs. We present in this chapter several algorithms that address the challenges inherit to these networks, focusing on solutions that address performance and scalability improvement. In fact, WMNs have been extensively addressed within the research community over the past decade, given their potential and commercial interest. Target scenarios vary from power energy efficiency to security or multicast. These however are not within the scope of this work. Our target scenario comprises scalability and performance, so as the solutions presented in this chapter.

We divided the broad range of schemes into three major categories: Channel Assignment on Section 2.1, Routing, Section 2.2 and Joint Routing and Channel Assignment on Section 2.3. To each of the solutions we address the implemented methodology, with a critical view in order to unlock the full potential of a new scheme. Hence at the end of each Section we present a brief discussion. Furthermore in Appendix A we provide a summary table to each category.

2.1. CHANNEL ASSIGNMENT

The performance of a MRMC network is largely affected by co-channel interference arising from multiple simultaneous transmissions from neighboring links. CA strategies aim to

minimize such interference by mapping the available channels to links while ensuring connectivity. The CA problem is known to be NP-hard which means that there is no known optimized solution.

2.1.1. MESH BASED TRAFFIC AND INTERFERENCE AWARE CHANNEL ASSIGNMENT (MESTIC)

Authors in [5] propose a static centralized greedy channel assignment scheme based on a ranking function. Each node maintains a default channel, for control purposes and also to prevent flow disruptions, bounded to a specific NIC. The CA algorithm runs according to the priority calculated by the ranking functions. This function is used to prioritize nodes based on traffic demands, distance measured in number of hops from the gateway and the number on radio interfaces. Rankings are calculated as shown in (2)

$$Rank\ (node) = \frac{Aggregated\ Traffic\ (node)}{\min hops\ from\ GW\ (node) * number\ of\ NICs\ (node)} \quad (2)$$

After all nodes have been ranked, the algorithm starts visiting each one in decreasing order of the raking function. The gateway node will be first, then on each node the algorithm first checks if it shares a link with an already visited node and if so assigns the same channel (remember that it uses a fixed topology), if not the NIC that relays more traffic is assigned first with the least used channel and the same happens if no links are associated with the NIC.

2.1.2. INTERFERENCE AND TRAFFIC AWARE CHANNEL ASSIGNMENT (ITACA)

ITACA [6] is a centralized hierarchical channel assignment proposal focused on the deployment of access networks and thus assuming that most traffic is directed to the gateway. Its main objective is to enable traffic flows over low interference and high capacity links by binding channels to nodes in such a way to reduce both intra and inter-flow interference. A Channel Assignment (CA) module located at the gateway is the central node that manages the channel distribution throughout the network.

It defines two different scenarios in a traffic aware scheme, either assuming that the traffic is homogeneously distributed among all nodes or congested. Initially, all nodes must be first operating in a common channel collecting information related to their distance from the gateway, link quality and estimated interference. The CA module must be informed about the number of nodes, its distance (in terms of hop-count) and the number of radios on each

node. Channels are then sectioned in default and non-default by the respective Default Channel Selection function and Non-default Channel Selection function. Both functions are recalled after a certain period of time (greater for common channel function) and in case quality decreases under a threshold value for a certain period of time.

To model interference between nodes and to select the best channel in a multi-radio scenario the conflict graph is extended to account for multiple interfaces with the Multi-Radio Conflict Graph (MCG). The MCG is built using node position, transmission range and carrier sensing range. Channels are ranked according to: $R_c = \frac{\sum_{i=1}^{n} Rank_c^i}{n}$ and the channel with the smallest R_c is selected as the default channel. Non-default channels are also assigned by the CA module using the Breadth-First Search algorithm (BFS-CA) giving high priority to nodes located closer to the gateway and with better delay figures.

This approach considers that traffic flows are constant along the network. To account for traffic aggregation, a Coefficient Variation (Cv) is defined as the ratio between the standard deviation of aggregated traffic crossing each link to its mean value. If the value Cv is higher than the threshold the channels are assigned starting from the gateway and selecting first links with higher value of aggregated traffic otherwise if value is lower than the threshold then only interference is considered.

To respond to topology changes the assigned channels can be changed by the CA module. A radio broadcasts INTERFACE-DOWN message in order to announce neighbors that the interface is inactive. Receiving this message all neighbors will delete the corresponding entry from the ARP table.

Evaluations available are based on simulation and real-world. The number of radios may vary, the only constrain is that the at least one radio must operate on the default channel when the channel assignment procedure is running. This is a centralized solution with a control dedicated NIC.

2.1.3. ADAPTIVE DYNAMIC CHANNEL ALLOCATION (ADCA) AND INTERFERENCE AND CONGESTION AWARE ROUTING (ICAR)

ADCA [7] is a hybrid multi radio multi-channel protocol that combines both the advantages of static and dynamic channel assignment. Each node on the network has at least two radios, one dynamic and the others static. The main purpose of static interfaces is to maximize

throughput from end-users to the wired infrastructure as dynamic interfaces increase network connectivity and help the network adapt to variable loads.

Channel allocation of static interfaces is based on *hyacinth* where a load balanced tree is constructed for each gateway giving high priority, and so less congested links, to nodes located closer to the gateways. Dynamic interfaces communicate with each other when they have data to transmit. In ADCA time is divided into fixed length intervals and each interval is further split into control (20ms) and data interval (80ms). In the control interval all nodes switch to the same default channel and negotiate the channels used in the data interval to transmit and receive. To minimize interference with static interfaces, the control channel for dynamic interfaces is excluded from the channel list of the static ones. Each dynamic interface maintains a data queue for each neighbor. During the control interval ADCA considers not only throughput but also the amount of time a queue as not been served to prevent starvation.

ADCA is able to negotiate a common channel between two or more nodes in each interval when traffic is below a saturation point. This negotiation process continues until the queue reaches a threshold value and an example scenario depicted in Figure 4.

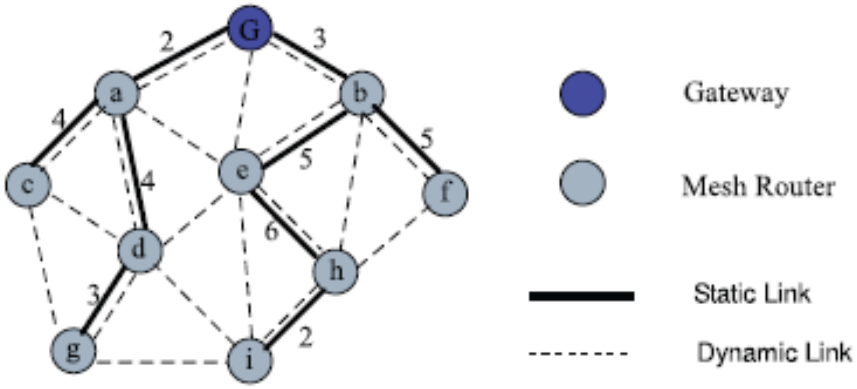


Figure 4 – ADCA Architecture [7]

The authors also propose a new routing metric called ICAR. Each link is categorized into three states: "congested" for links highly congested through which is not desirable to route additional traffic, "median" for links with high congestion but still able to route additional traffic and "low" denotes links with very low congestion and thus preferred to be used. This categorization is used to weight the cost of a link. Link states are inferred from the queue

length (the longer the queue the higher the congestion). Interference in each dynamic link is estimated by maintaining a channel usage history in each node and updated periodically in the routing agents.

The threshold value tries to keep delays low, but the presented results show that data rates above 3250 Kbps result in high delay. Also the negotiation process at low data proves to be inefficient below 1500 Kbps

2.1.4. IMPROVED GRAVITATIONAL SEARCH ALGORITHM (IGSA)

In this solution [8] authors used an improved version of the gravitational search algorithm to solve the problem of channel assignment in mesh networks focused on the reduction of interference and increase throughput. The gravitational search algorithm is inspired by Newton laws of gravity and motion. The quality of a solution is equivalent to the mass of an object so that good solutions attract other solutions towards them.

A disruption operator is used to gradually reduce search space and hence converge the algorithm to the best solution by eliminating others that diverge the most from the best. A discrete local search (DLS) operator is added to the gravitational search algorithm to enhance the performance and find the best solution among good solutions. The DLS operator is applied to the best solution of the system generating a random number between zero and one comparing it with $P(t)=t/(2T)$ where t is the iteration counter and T the total number of iterations. If the random number is less than P(t) then the DLS is applied to the best solution of the system.

The DLS randomly selects a node and changes some of the channel allocated to the node. In IGSA the search is more accurate so that when the algorithm is converging to a solution the DLS operator is applied with higher probability to find better solutions.

IGSA searches for the optimum CA based on the principle that since the mesh topology is static its characteristics are also stable. The found channel assignment is the static for a certain (long) period of time. Both of these characteristics make it hard to adapt and quickly react to eventual oscillations. In simulations the authors considered full connectivity or topology preservation.

2.1.5. CLUSTER CHANNEL ASSIGNMENT (CCA)

The authors in CCA [9] propose a cluster based channel assignment aimed at the reduce of interference and thus the increase of network capacity. The mesh network is constituted by a set of mesh routers belonging to one of the *C* clusters. Clustering is performed using the *Highest Connectivity Cluster* algorithm (HCC) where the node with highest number of neighboring nodes is denoted the *Cluster Head* (CH). Additionally from all the elected CH one will be the *Head of the Cluster Heads* (HCH) and the cluster will then be promoted to *Central Cluster*. Channel assignment in CCA algorithm is performed in three stages: Channel division and selection, Channel re-assignment, Cluster nodes CA.

The HCH disjointly assigns a set of channels to clusters, which means that a channel assigned to a cluster cannot be assigned to another cluster. Having the set of channels allotted, each CH starts by assigning a common channel to nodes that belong to its cluster. Border nodes on neighbor clusters might negotiate a common channel to make inter-cluster communication feasible. The remaining channels are assigned to the available interfaces starting by the least used one. Additionally channel reuse is allowed between nodes that are two clusters apart from each other.

To assign channels to a node, each CH must at first gather information about the number of NICs, neighbors, channels assigned to links within its cluster and also the SC connectivity graph. This can be achieved during the network discovery process using a SC. A node is assigned a channel by its corresponding CH accordingly to its number of NIC and channels used by the cluster.

Cluster solutions may be a good option for large scale mesh networks, however in small size environments clustering doesn't seem feasible. Also the authors despised traffic load in channel allocation which can result in interference.

2.1.6. CLUSTER-BASED MULTIPATH TOPOLOGY CONTROL AND CHANNEL ASSIGNMENT (COMTAC)

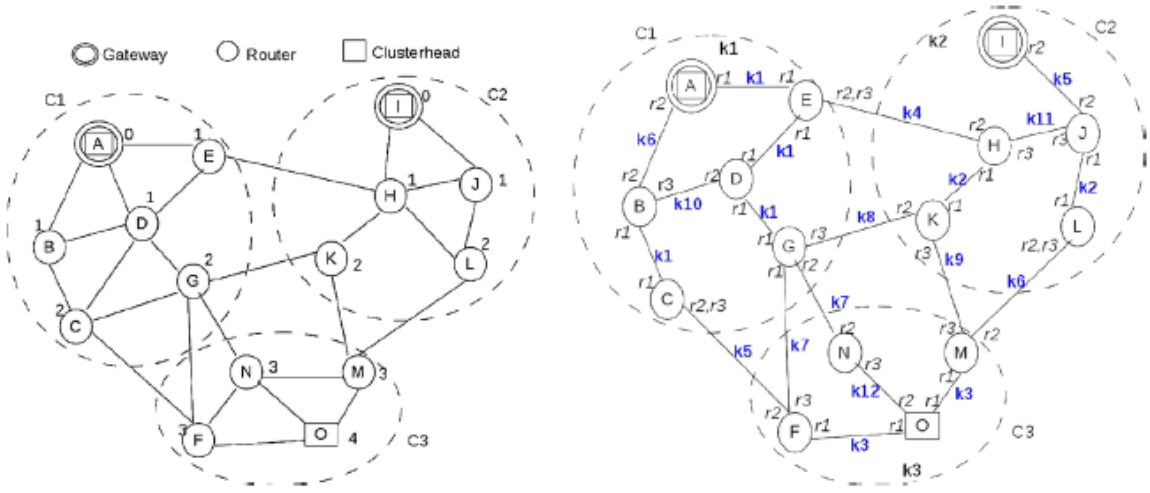
The authors in [10] present a cluster-based multipath topology control and channel assignment scheme named COMTAC. It is a hierarchical cluster based channel assignment scheme where clusters are created based on hop-distance and each node within each cluster binds one NIC to a default channel. Additionally nodes that border multiple clusters allot a

second interface to the default channel of the cluster with highest priority to enable inter-cluster communication.

The topology creation process is separated from channel assignment to minimize flow disruptions. Starting with topology creation, gateway nodes are the first CH and create the respective clusters comprising all nodes connected to them. The algorithm then iterates over the clustering until all nodes in each cluster have at most *R* hops distance from the CH choosing as new CHs nodes located further (in terms of hop count). Nodes are assigned to the newly formed cluster if the distance from the new CH is lesser than the distance from the current.

Within each cluster the default interface of each node is assigned to the default channel. The selection of the default channel is controlled by the CH of each cluster. It selects random few nodes redirecting all traffic flows to the default channel for a period of time and using the non-default interfaces to measure traffic flows and channel quality (using soft metrics). This information is gathered by the CH that computes the cost of each channel and changes if it represents the one with the least cost. Non-default interfaces are assigned with channels based on the average queue length with interference domain assuming that longer queues are the result of higher interference. The information about queue lengths experienced on all the assigned channels is transmitted periodically by each node (including border nodes) to the CH that computes the new channel assignment for all cluster nodes that can be either using non-overlapping or partially overlapping channels. On the assignment, priority is given first to border nodes according to neighbor cluster priority and only performs if the neighboring nodes are bounded to it and not the other way and secondly to interfaces bounded to more neighbors inside the cluster.

Further improvements to the topology are achieved through a process called the spanner construction that takes as inputs link and path interference. In result some links may disappear, as shown in Figure 5.b) in the process and be substituted by a two hop path if the interference of the path is smaller than two times the interference of the link.



(a) COMTAC Clustering (p=2)

(b) COMTAC Spanner, IA and CA

Figure 5 – COMTAC Clusters and CA

2.1.7. CLUSTER BASED CHANNEL ASSIGNMENT (CBCA)

CBCA [11] authors propose a hierarchical cluster based channel assignment algorithm to minimize co-channel interference yet retaining network topology. Channel assignment is performed in a distributed manner over the clusters. Network is divided into *C* clusters each one composed by an exclusive set of nodes.

Clustering is performed based on *Connectivity and ID* (CONID), which uses connectivity and id as primary and secondary keys respectively, and is focused on the reduce of the number of clusters and border nodes. The main reason is to reduce of broadcast messages that are only issued by CH and border nodes. CBCA uses the same three stage approach as CCA to perform: clustering, neighbor to interface binding and channel to interface assignment.

Clustering is a centralized operation performed at start-up by a designated gateway node to find a fixed topology that does not change too much through the rest of the network operations. This operation takes as input the connectivity graph *G*, distance in terms of hop-count and the expected cluster radius outputting a hierarchical cluster set, where clusters closer to the gateway are given higher priority in channel assignment. Each node is informed about its cluster and the respective CH. The CH is elected among the nodes within a cluster as the one closer to the gateway and with most neighbors.

Neighbor to interface binding is performed in two steps. First the interfaces in border nodes are allocated to provide intra and inter-cluster connectivity, using one per neighboring cluster (if available) or reusing interfaces assigned to inter-cluster communication if the number of neighboring cluster outnumbers the available non-default interfaces. Secondly interfaces are assigned to neighbor nodes that belong to the same cluster starting first by nodes located closer to the gateway.

Channel assignment prioritizes interfaces used for inter-cluster communication. Border nodes may decide if the interface is greedily assign or if the node waits until a bounded neighbor picks a channel, this prevents channel oscillation problems. This decision is based on cluster id, the neighbor node with the lowest id gets to decide. A common channel is assigned to all nodes in the network to satisfy topology preservation. To minimize the ripple effect of changing a channel in the network border nodes use one specific interface to communicate with nodes from a different cluster, also interfaces are used to communicate with neighbors within a hop count threshold. This process is depicted in Figure 6.

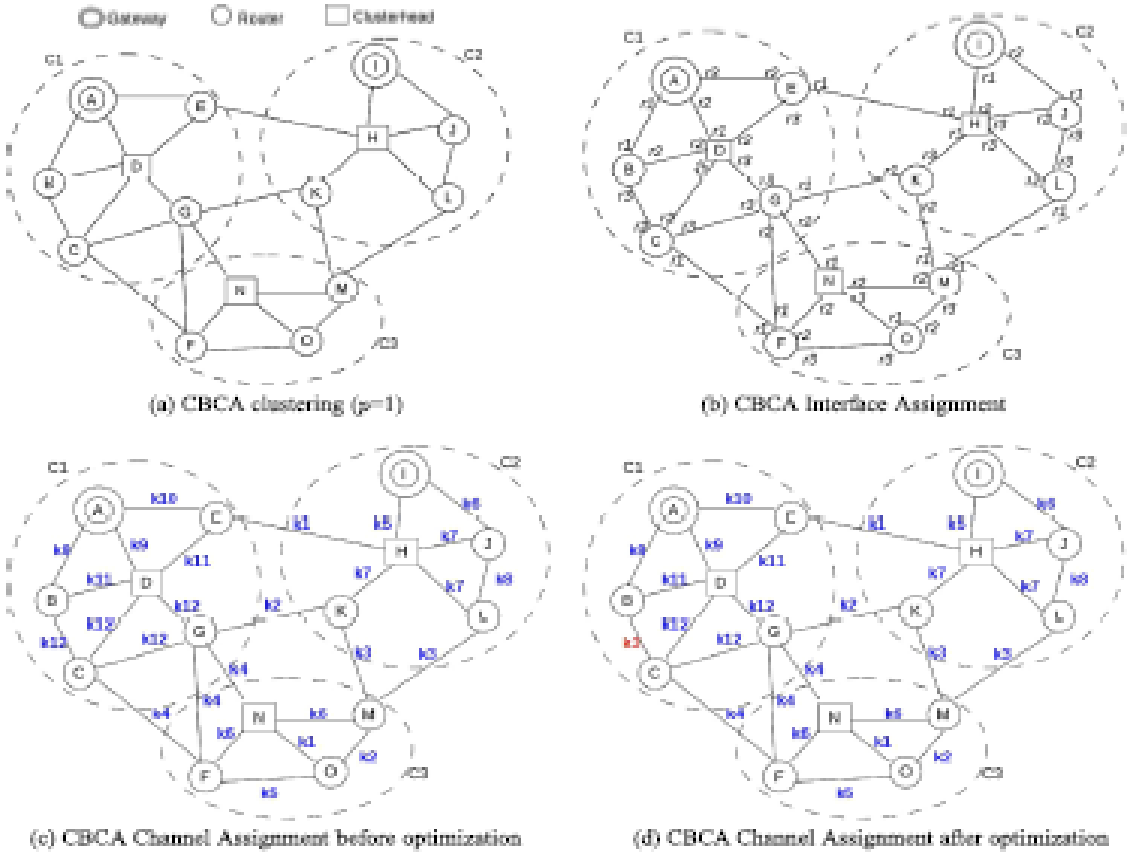


Figure 6 – CBCA clustering and CA [11]

2.1.8. DISCRETE PARTICLE SWARM OPTIMIZATION (DPSO-CA)

Hongju et al. [12] propose a channel assignment scheme with topology preservation based on the discrete particle swarm caller Discrete Particle Swarm Optimization (DPSO-CA). Discrete particle swarm optimization is an iterative process in which a particle, i.e. the network channel assignment scheme, adjusts its position, i.e. channel assignment, accordingly to its experience and the experience of its neighbors. The position update process borrows the concept from genetic-based algorithms, since channels are discrete variables, using crossover operations to improve positions based on local and global best value and mutation operation to change the position randomly and avoid the local optimization problem. The whole algorithm works around the solution that corresponds to the global minimum interference.

2.1.9. PARTIALLY OVERLAPPED CHANNEL ASSIGNMENT (POCA)

A traffic-irrelevant with partially overlapped channel assignment is proposed in [13] short named POCA. Authors propose a theoretical calculation to relate the interference range and channel separation. The interference range when using partially overlapping channels is reduced by a factor that corresponds to the extent of the overlap between both when compared with co-channel interference range. The parameter that measures the overlap extension also considers the signal power distribution across the frequency spectrum

The algorithm is divided into two distinct phases. First Neighbor-to-Interface binding allows each to node fairly distribute its interfaces among neighbors. Interface to channel binding is performed in the second phase considering both network topology and interference levels. In this phase links are ordered in ascending order of expected interference levels and ranked to break any ties using the two endpoint nodes degree as well as the average distance to gateway. Each link is then visited and assigned the channel that represents the least degree of interference on the network, calculated for every candidate as the sum of the interference between a link and all the other links that have already been assigned a channel.

Traffic irrelevant schemes are operated before any traffic flow exists on the network, so the channel assignment obtained in these conditions will be maintained through the rest of the network operation and no response will be given to any performance decay. Even if not implemented the authors describe the procedure of a distributed scheme using periodic broadcasts from the gateway node advertising its presence, so that nodes can estimate its

distance in terms of hop-count, and information exchange messages broadcasted by each node with the neighborhood.

2.1.10. CONNECTED LOW INTERFERENCE CHANNEL ASSIGNMENT (CLICA)

In [14] authors developed the Connected Low Interference Channel Assignment (CLICA), a heuristic algorithm that greedily assigns channels to nodes according to its priority, based on the connectivity (G) and conflict graph (G') models and ILP formulation. The priority is used to establish the order of channel assignment and can be based on distance in terms of hop-count to the gateway or traffic load. Despite the established priority the algorithm may override the order of channel assignment to retain network connectivity. To reduce interference CLICA also considers the use of fewer NICs per node.

The algorithm is recursive and not iterative, after the channel assignment procedure completes nodes are no longer visited. It takes as input the connectivity and conflict graph to create a vector in decreasing order of priority for the selected parameter and greedily selects the channel that locally minimizes interference. Also the nodes with least interfaces can be given higher priority. After selecting the node with highest priority the algorithm visits the associated links assigning the color that minimizes the link conflict weight (sum of weights of edges incident on the corresponding vertex in the conflict graph) for the link in use.

2.1.11. CHANNEL ASSIGNMENT WITH MULTIPLE FACTOR CONSIDERATIONS (CAMF)

Jenn-Wei Lin et al. [15] propose an on-demand hierarchical channel assignment scheme aiming at multicast transmissions and using both non-overlapping and partially-overlapping channels. The hierarchy is defined based on the number of members dependent on transmissions of a multicast path on the multicast tree, they call this parameter forwarding weight. For control purposes each node uses a specific network card tuned to a default channel.

The forwarding weight parameter is estimated bottom-up on the multicast tree adding the number of children on each intermediate node and additionally the number of receivers using the control channel. Also through this channel nodes exchange interference-announcing and acknowledgment messages with the to the two-hop neighbors to build a pessimistic interfering node set considering every node within interference range.

When a node wants to select a channel for broadcast transmission it will use the information present in the interfering node set and also consider the channels used by all parent nodes given their higher priority disregarding channels used by children nodes. The selected channel is the one with least interference and this information must be broadcasted to all neighbors. To choose among non-overlapping or partially overlapping each node must weigh the assignment considering the distance to a node using the same channel Nodes also adjust their contention window in two situations: according to their priority, so nodes with higher path forwarding weight get a smaller contention window, and to prevent frequent channel re-assignments due to variations of the forwarding weight parameter.

The whole process works on-demand initiating and terminating routes according the multicast receivers and their mobility. Each path is only maintained as long as receiver demands it.

2.1.12. DISCUSSION

Solutions like MESTIC, ITACA, ADCA, COMTAC: CBCA, DPSO-CA require both a control channel and NIC thus not taking full advantage of multiple radios and multiple channels as the full potential of radios and spectrum is reserved only for control purposes.

Most centralized protocols like MESTIC, ITACA, CCA, CBCA, DPSO-CA, POCA and CLICA require traffic to converge to a central entity that then proceed to channel assignment leading to increased delay and computational needs. Except for CLICA, all solutions require a control dedicated channel which may also become the main bottleneck.

CCA, COMTAC and CBCA are similar approaches. CCA is the most simple, as it chooses a CHH that will randomly assign channels to cluster. This approach despises important inputs like traffic load or interference levels, so channel assignment may not be optimum. COMTAC aims to reduce interference in a hierarchical topology, and it reserves two interfaces for broadcasting within the cluster and another for inter-cluster communications. CBCA is also based on a hierarchical topology but computationally heavier than COMTAC as is used both a centralized clustering algorithm and a distributed CA. As already mentioned cluster approaches fit well on large networks but are more likely not to bring much improvement in small size environments.

Heuristic and genetic-based approaches like IGSA and DPSO-CA are most targeted at static deployments. The output of either algorithm provides near optimal solution to the problem they try to solve, however these are computationally heavier than other proposals and this problem scales as network grows.

Using partially overlapped channels it is possible take advantage of full channel spectrum with controlled interference, like in POCA. However the procedures required to make it efficient are computationally heavy and require either a central entity to perform channel assignment or heavy control overhead, resulting in increased latency and delay.

ADCA uses static interfaces to improve throughput from mesh clients to the cable system and dynamic interfaces to enhance connectivity and respond to load variations alternating in a TDMA style between control and data intervals. When data rates exceed a certain value, channel negotiations occur among neighbors using the dynamic interfaces and control intervals may overlap data intervals. During the negotiation period queues line up for each neighbor then priority is given to longer queues. If queues exceed a threshold values the channel negotiation stops. This whole process results in unfair load distribution and starvation may occur for node to which queues kept low for long periods of time.

ITACA accounts for traffic flows by considering the overall nonconformity of flows over a threshold and value reassigning channels to all network. The channel reassignment process may result in flow disruptions and high latency for a period of time that increases as the network scales.

Although multicast protocols for mesh networks are out of the scope of this dissertation CAMF is presented as a way to acknowledge the existence of other solutions.

2.2. ROUTING

The impact of routing strategies has a substantial effect on network's performance. Given the source and destination of a packet the main objective of routing strategies is to find low overhead paths to deliver the packet in shortest time.

2.2.1. WI-FI NETWORK INFRASTRUCTURE EXTENSION (WIFIX)

WiFIX [16] is a distributed routing protocol based on IEEE MAC learning bridges (802.1d) that relies on a single message to establish links and provide self-organization in a

hierarchical spanning tree architecture. It is focused on extending the wired infrastructure, so most traffic is relayed from mesh clients to the GW.

WiFIX extends the ad-hoc frame (Figure 7) header to the 4 MAC address format (Figure 8) without modifying the standard, using a tunneling mechanism called Ethernet over 802.11 (Eo11). Along the path to the gateway frames are encapsulated and decapsulated. At each hop, link source and destination address are added to the MAC frame, preserving the original source and destination in the Eo11 header.

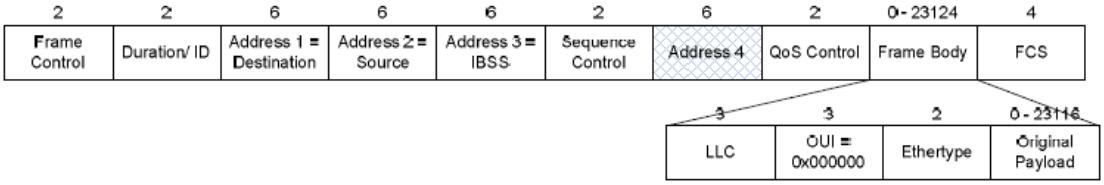


Figure 7 – 802.11 frame format [16]

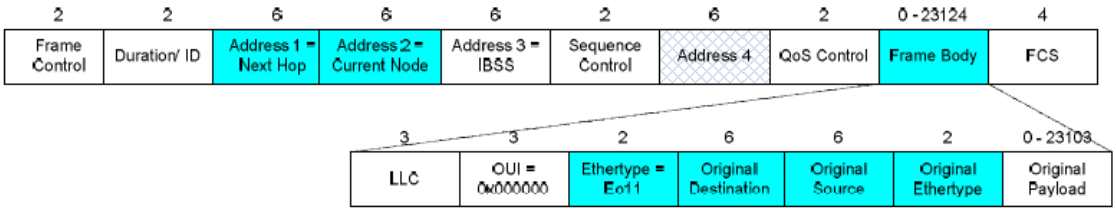


Figure 8 – Eo11 frame format [16]

Routing is performed over layer-2. The abstraction to upper layers is provided by Eo11 tunneling mechanism with each tunnel endpoint behaving like an Ethernet port from upper layers point of view. This also enables that a virtual learning bridge is implemented at each node where connected stations hang on bridge's ports. Upon receiving a frame the source address is bounded to the corresponding bridge port.

A mechanism called Active Topology Creation and Maintenance (ATCM) is used to create an active tree topology rooted at the master access point. A Topology Refresh (TR) message is sent periodically to announce the Master AP and notify the upstream nodes that a given child is at its parent node in the tree.

2.2.2. ADAPTIVE STATE BASED MULTI-PATH ROUTING PROTOCOL (ASMRP)

Authors in [17] propose a routing protocol with the purpose of providing multiple paths to the gateway and opportunistically use them according to congestion levels.

It comprises a hierarchical architecture where mesh backbone levels are determined by the number of hops to the gateway. Topology creation and maintenance is performed using a set of messages. HELLO messages are sent by gateways to advertise their presence and inform neighbors about average load and link capacity which MAPs will use to update routing tables. MAPs also send HELLO messages containing a set of routes to reach the GW along with their performance metrics. They also send unicast PARENT-NOTIFY messages to advertise their presence to parent nodes and CHILD-NOTIFY to inform upstream nodes of a new child at their parenthood.

Routing is performed considering congestion level measured at the routing layer, with respect to the queue size to each next hop candidate over a certain period of time. The logic is that longer queues over long periods of time increase packet loss probability and so should be avoided. The full-path is chosen by the source node, although intermediate MAPs may opt to transmit through a different path if the route is considered unstable. The instability of a link is determined by a state machine, at each node, that determines promising neighbors and reliable routes.

Each radio is equipped with three radios. Sending and receiving traffic occurs in a separate interfaces, in an attempt to close to the full-duplex behavior, while another is used sending and receiving broadcast messages. The receiving interface is tuned to a fixed channel while the transmitting interface switches channels to the ones used by next hop neighbors with respect to the Neighbor Channel Table.

2.2.3. CHANNEL AWARE OPPORTUNISTIC ROUTING (CAOR)

The authors of CAOR [18] explore the concept of Opportunistic Routing (OR) to prove the concept of the also proposed routing metric Multi-Channel Expected Anypath Transmission Time (MEATT). In OR the broadcast nature of Wireless transmissions is explored so when a node hears a transmission it may forward it to the destination. So as long as there is always one forwarder closer to the destination the data can move forward. The candidate forwarders constitute the forwarder set. Figure 9 depicts the OR concept.

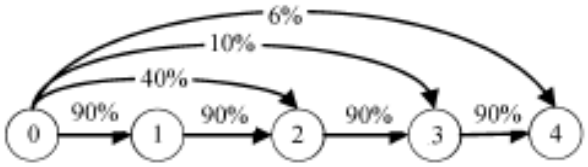


Figure 9 – Concept of OR with packet delivery ratio

The goal of MEATT metric is to select the best forwarders set among all neighbors set and to select a radio to interact with each neighbor. It considers the expected transmission time and adds a tunable parameter in order to avoid consecutive hops operating on identical channels reducing interference and enhancing throughput. The cost of all candidates is taken into account.

CAOR is a distributed proactive routing algorithm that based on MEATT metric estimates the upper-bound to the shortest multi-channel anypath from all nodes to a destination. Each node has an independent estimate of MEATT for each channel to a destination that along with candidate forwarder set and channel constitute the routing table. Like in distributed Bellman-Ford routing algorithm, after weighting metrics to a destination, nodes broadcast the *path vector* that contains relevant information to neighboring nodes in all channels, like destination, MEATT weights and channel, that will use it to update routing tables.

Authors consider CAOR as a synchronous or asynchronous protocol. In synchronous, the operation time is partitioned in finite regular intervals that nodes use to estimate metrics and broadcast the *path vector* if any updates to the routing table sorted. In asynchronous, the operation is ensured by periodic broadcasts of *path vector*.

2.2.4. CROSS-LAYER QOS-AWARE OLSR (CLQ-OLSR)

CLQ-OLSR [19] is a routing protocol that implements QoS in the OLSR protocol to support multimedia and IP traffic. Like OLSR, CLQ-OLSR is a proactive protocol where nodes nominate a group of special nodes, at 1-hop distance, as the Multi-Point Relay nodes (MPRs). MPR nodes declare the state of the subset of links to its MPR selectors. Control messages are periodically sent which can be from three different types: HELLO, Topology Control (TC) and Multiple Interface Declaration MID. HELLO messages perform the task of link sensing, neighbor detection and MPR signaling. TC messages carry the topology information as the advertisement of the link state. MID messages are used to notify the

presence of multiple interfaces and the binding relations on a node. It uses shortest path algorithm to perform path calculation when any changes in topology, interfaces or links are detected. It is an iterative process that begins with the deletion of all items.

This protocol considers two different topologies physical and logical. The physical topology is maintained by M-OLSR protocol and is used to forward best-effort traffic and disseminate HELLO, TC and MID messages. The logical topology is used to forward real-time traffic and considers the values obtained from M-OLSR to build a logical topology accordingly to the session QoS parameters. To guarantee QoS nodes continuously monitor the environment sensing on a SC and adding up the idle time over a period T. The idle time is said to be the time in which the medium is free, no nodes in interference range are transmitting. The available bandwidth is the percentage of time multiplied to the total available bandwidth. This values are then used in the *channel selection index* to identify the best channel to configure the current interface.

Best-effort traffic is simply forwarded to the destination on hop-by-hop through the physical network. When real-time traffic is to be forward a logical mesh topology is generated based on the physical topology and the available bandwidth obtained from M-OLSR. The source node starts by creating a logical mesh topology based on the physical shortest path. Then the corresponding available bandwidth is computed considering the lowest bandwidth available in the constituent links. To balance the relation between path bandwidth and path cost a set of constrains are imposed to make the chosen path the path with: hop count less than a certain value, highest bandwidth available, minimum physical hops and found first.

Packets on logical paths are encapsulated by a logical routing module (LRM) which indicates the address sequence of intermediate nodes on the physical path is maintained by M-OLSR module PRM and forwarded by a forwarding module.

The number of interfaces per node is not specified but variable with one interface set to accommodate the modified OLSR protocol and deliver best-effort traffic. The remaining ones are responsible to deliver multimedia real-time traffic. Routing performed on layer 3.

2.2.5. MULTI-CHANNEL DUAL-RADIO PROTOCOL (MCDR)

A routing protocol with topology discovery and maintenance is proposed in [20] named MCDR. MCDR uses WCETT-LB metric to choose between channel diversified routes and

the DIM MAC protocol to access the medium. Dual-radio nodes are used where one fixed interface is used for data reception and another switchable interface to distribute traffic to neighbors. The protocol is divided in two parts: routing criterion design and routing discovery and maintenance.

Routing criterion and design considers a set of parameters to greedily establish the best routes up to a certain destination. Because the DIM protocol uses a switchable NIC to distribute the traffic load as it increases the channel switching time is weighted when finding the best route. The link quality is inferred from detection packets and the respective acknowledgements sent periodically. Another used parameter is the node load degree expressed as the queue size over the sending rate on a specific path. Finally the authors used WCETT-LB metric to weight a specific path. This metric is an extension of WCETT with a load-balancing component to provide load distribution and avoid congestion.

Nodes broadcast RREQ messages when no entry to destination is found, intermediate nodes will rebroadcast these messages until the destination is reached. Upon receiving these requests the destination node will select the best route for transmission and send a unicast packet to the source route. The source node will attach the route to the head of each datagram. During the transmission and in case of link failure intermediate nodes will send RRER packets to the destination node and delete this entry from the routing cache, all nodes that receive this message also delete the entry and the source node must then start a new route discovery process.

MCDR is similar to OLSR and performs routing over layer 3. As discussed before performing routing on layer 2 is preferred.

2.2.6. BETTER APPROACH TO MOBILE AD-HOC NETWORKING-ADVANCED (BATMAN-ADV)

The Better Approach To Mobile Ad-hoc Networking advanced (BATMAN-adv) protocol [21] [22] is a proactive routing protocol that extends the original BATMAN protocol to Layer 2.5. It works by periodically flooding the network with Originator Messages (OGM) and using these same messages to update the routing tables. When a node sends an OGM packet it will be rebroadcasted by its neighbors, the objective is that it reaches the whole network and all nodes know how to reach the originator node. Every node in the network is

aware of the others and knows which is the best next-hop to reach them. They do not maintain the entire path in cache but only destination and best next hop.

The OGM message is also used to measure the transmission quality (TQ) as the number of transmitted packets over the received ones counting the number of rebroadcasts received for a given sequence number. The sequence number is maintained for each neighbor.

BATMAN-adv gateway nodes advertise themselves on the routing protocol level and mesh nodes select their default gateway as best gateway in terms of TQ. Each node supports multiple interfaces in two modes: Interface alternating and Interface bonding. In interface alternating nodes never forward a packet on the same interface on which it was received but possibly on a different one if it avoids performance degradation. Interface Bonding uses each NIC in a round-robin fashion to send the packets. Bottleneck interfaces will limit the usable bandwidth and also worst quality links will affect the entire transmission.

2.2.7. DISCUSSION

Routing on layer 2 is preferred because as no lookups to tables are performed a packet is sent faster than when perform on layer 3. Additionally the abstraction it provides to upper layers it's an asset, especially to layer 3 where major changes occur due to the replacement of IPv6 over IPv4.

ASMRP uses a mechanism to balance the overall network load and improve performance by keeping nodes aware of congestion on the path to the gateway and routing through less congested links. Also they consider the use of different performance parameters according to different types of applications. This is a good approach since traffic might be forwarded according to QoS demands of each application, providing an even greater balance. Although, an excessive amount of control messages with high payload are used which can lead to high latency and high convergence time during initialization phase. Also each node must store a great amount of information such as channel table and each possible route along with each link performance. These problems scale as network grows. Channel switching times must also be accounted, since this operation is not completed in short time without modifying the standards. This is also the main drawback of BATMAN-adv. WiFIX on the other hand disregards traffic load on links that compose the path to the gateway. Although this may prevent frequent path oscillations the network won't react around congested links.

The opportunistic routing takes advantage of the broadcast nature in wireless environments to create hop shortcuts and thus speed transmissions. However a multi-channel deployment may make the opportunistic routing inefficient as packets will only be forwarded by nodes tuned to the same channel, so a compromise between interference and opportunistic routing must exist.

2.3. JOINT CHANNEL ASSIGNMENT AND ROUTING

JCAR solutions are considered to be NP-complete [23], given the already stated interdependency between routing and channel assignment. This means that a solution to one problem is easily found if a solution exists to the other problem. Hence, the proposed solutions presented in this chapter give priority to either CA or Routing to then address the other.

2.3.1. ROUTING OVER A MULTI RADIO ACCESS NETWORK (ROMA)

ROMA [24] is a hierarchical and distributed protocol that performs routing and channel assignment in a dual radio-network to achieve good end to end throughput. Each gateway creates its own set of non-overlapping channels to assign to nodes hierarchically below it. Thus, inter-path interference is reduced in a multi-gateway scenario as multiple gateways try to use different channels. Each node along the path is equipped with two interfaces operating on two different bands (802.11a and 802.11b/g) in order to guarantee that both could operate simultaneously with no cross-channel interference due to close range operation. The architecture is shown in Figure 10.

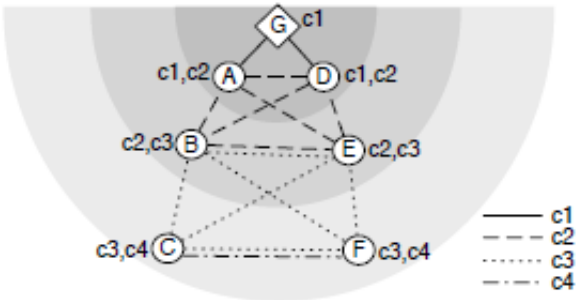


Figure 10 – ROMA architecture [24]

The gateway scans the environment for available channels and also chooses a random sequence of non-overlapping channels. This sequence is part of the route advertisement and each node determines its channel assignment based on the best way to the gateway and the

corresponding channel sequence. With this approach it's possible to eliminate the use of a common channel along the tree while still ensuring that the routing protocol can find low overhead paths. The assignment of the same channel to nodes at the same level also provides multiple paths to the gateway and since ultimately all nodes compete with each other at the first hop of the gateway there is little performance cost.

To choose the most promising link in this distributed scenario two types of information are included in route advertising messages: Neighbors channel assignment and their gateway path metrics. The metric used in ROMA considers both the Expected Transmission Time (ETT) and the observed external load so that longer paths with less interference might be chosen over shorter paths. Upon finding its best gateway path each node switches its interfaces to the assigned channels, also called home channels, and continuously monitors each neighboring link. This monitoring occurs periodically with the node snooping the medium to estimate external load and sending bursts of twenty probe packets to measure the delivery ratio.

ROMA gives a node the alternative to assign identical channels as its previous hop neighbor if it finds the link signal levels lead to a low quality link. Each node can also temporarily switch channels to try to discover better routing paths to the gateway, especially in a multi-gateway scenario.

2.3.2. LOAD AWARE CHANNEL ASSIGNMENT (LACA)

LACA [23] is a dynamic centralized channel assignment and routing approach which considers an estimation of the traffic load, network topology and the number of interfaces available on each node to assign channels to interfaces and define routes. Channel assignment and routing aim at the maximization of the *cross-section goodput* that is used as the evaluation metric. This parameter considers the expected load to perform bandwidth allocation on each link. Two channel assignment algorithms are proposed. The first one considers the network topology to perform channel assignment and is called *Neighbor Partitioning Scheme*. The second one exploits traffic load information to assign channels to nodes and it's called *Load-Aware Channel Assignment*.

Starting at a node the *Neighbor Partitioning Scheme* partitions its neighbors into groups and to each, one interface is assigned, thus the number of groups depends on the number of interfaces on each node. The process is repeated by its neighbors and then neighbors of

neighbors, always with respect to the previously formed groups, until all nodes on the network have completed group formation and we have a fully connected mesh. Then the least used channel on the neighborhood is assigned to the group.

The *Load Aware Channel Assignment* algorithm adds one more degree of improvement by further exploiting traffic load information. The traffic profile is traced considering the expected load on each link of a node. The algorithm iterates first over routing and channel assignment, initially disregarding link capacity and until it gets as close as possible to the expected load. This is called the *exploration phase*. At the end of each iteration, if some of the link loads are above the channel capacity the algorithm is repeated, visiting first links with higher expected load, until no further improvement is possible. The next phase, *convergence phase*, is very similar. The difference is that only the links that are below the expected capacity are recomputed.

The initial link load estimation is obtained from the expected loads from all paths that pass through that link over the total of paths between them. This whole process is depicted in Figure 11.

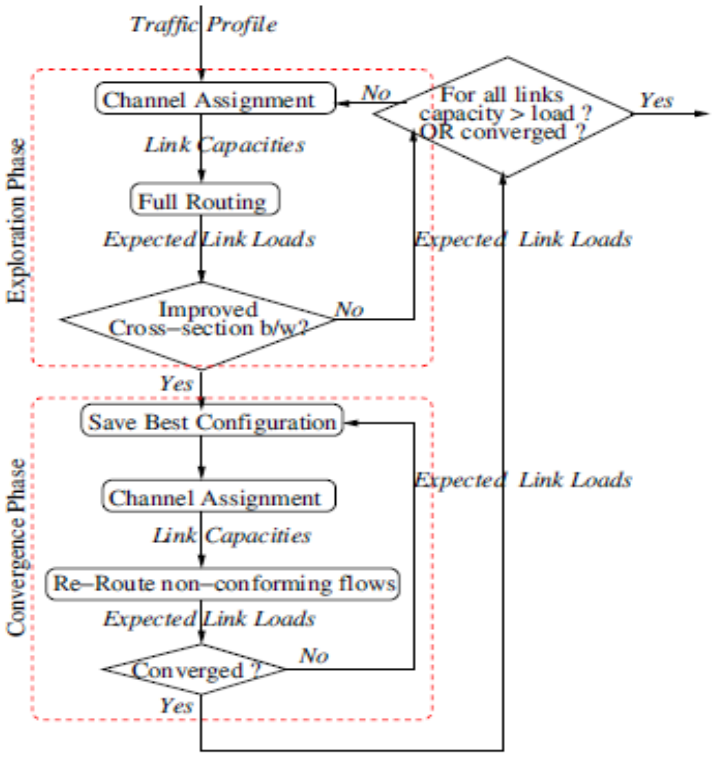


Figure 11 – Overall Iterations [6]

There are three possible cases of channel assignment. If the members in the channel list are fewer than the number of NICs then it's assigned the channel with the least degree of interference. If the number of members in the channel list matches the number of NICs in a node but is less in the other node then the channel with the least degree of interference from the first node list is chosen, assigned to the virtual link and then added to the second node list. If the number of entries in the channel list matches the number of NICs in both nodes it's picked the channel with the least degree of interference. In this case it´s also possible to choose a channel from each node and merge them into one channel. The combined degree of interference from the two picked channels is minimized.

2.3.3. HYACINTH

In [25] authors propose a spanning tree based architecture with its roots located at gateway nodes. It's a distributed channel assignment solution where local information is used to perform channel allocation.

The channel assignment goal is to bind the network interfaces to a radio channel and share that same channel to the communication range nodes. Each gateway node is connected to the wired network and acts as the root of a spanning tree which connect with each other through the wired network. By joining to multiple spanning trees each node of the WMN can distribute its load among them and also use one as a redundant link. Figure 12 shows hyacinth architecture.

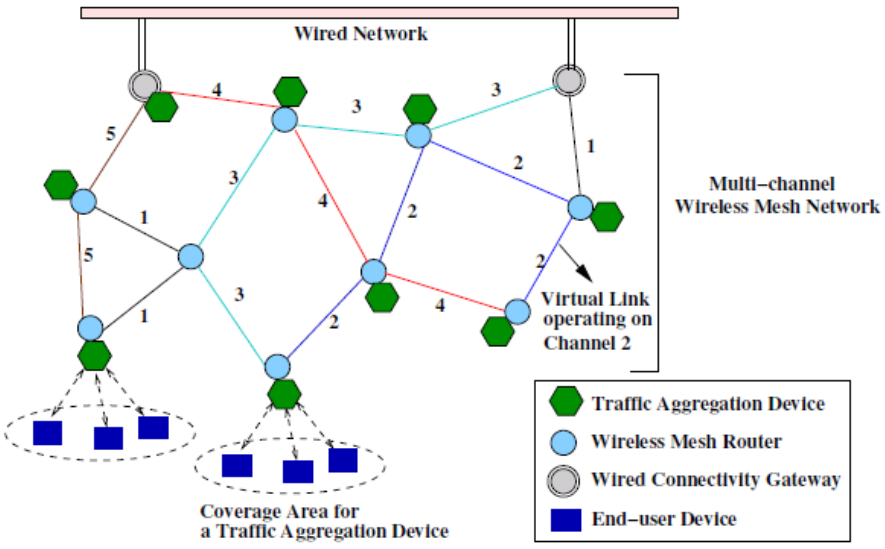


Figure 12 - Hyacinth architecture [25]

Control messages are evolved in topology creation as part of the Neighbor Discovery Protocol, HELLO messages are used to establish connections between neighboring nodes and ADVERTISE messages carry the metrics cost and are sent periodically. JOIN/LEAVE and ACCEPT/REJECT messages enable the establishment of parent/children relations. This messages are delivered over Internet Protocol (IP) multicast. The JOIN message propagates up to the gateway. It or any intermediate nodes can decide to send a ACCEPT or REJECT message to the new node if residual link is capacity is low (if it's a gateway) or capacity has decreased because other nodes have joined (if it's an intermediate node). Routing tables must be updated after ACCEPT and LEAVE messages are sent, to do so RT_ADD/RT_DEL are propagated all the way up to the root.

The metrics carried by ADVERTISE messages enable each node to determine the best path to the gateway, it considers both hop-count (number of hops to the gateway), gateway link capacity (total available bandwidth) and path capacity (minimum residual bandwidth on the path to the gateway).

The channel assignment problem is divided into two sub problems: neighbor-to-interface binding and channel-to-interface binding. In the first phase each node must determine which interface it uses to communicate with each neighbor. Some restrictions are applied, a node must fairly divide its NICs into UP-NICS, to communicate with parent nodes, and DOWN-NICS to communicate with the respective children, the exceptions are gateway nodes that only assign DOWN-NICS and farther nodes (may use more than a SC to improve bandwidth since they have lower priority on channel assignment). In the second phase channels are assigned to interfaces. The UP-NICs assignment is the responsibility of its parent (DOWN-NICs from parents and UP-NICs from children must match). To assign channels to DOWN-NICs a node continuously checks the usage of channels from neighbors within a maximum of 3-hop distance, exchanging CHNL_USAGE messages periodically. It then chooses a channel according to its level on the hierarchy.

2.3.4. JOINT ROUTING CHANNEL ASSIGNMENT (JRCA)

In[26] and [27] the authors propose Joint Routing Channel Assignment scheme. Both have the same name although differ in operation.

In [26] to obtain the initial channel assignment for the network the JRCA scheme first estimates channel load on the network using an existing channel assignment algorithm called

Maxflow–based Centralized Channel Assignment (MCCA). Two links are then created one with the MCCA algorithm and other with a bandwidth allocation equal to the bandwidth supported by the NIC. For every other link with no channel assignment, the residual demand is calculated, i.e., actual link demand minus the allocated demand. Once this operation finishes, links with non-zero residual demand are revisited in decreasing order and the least used channel are assigned.

[27] is a centralized hybrid approach that assumes a dedicated control NIC on each node tuned to a default channel through which nodes exchange Route Request (RREQ) and Route Reply (RREP) messages and all other interfaces switch between data channels. A combined backtracking with genetic-based algorithm in is used to perform channel assignment and reduce convergence time. To choose the best route a metric is considered taking into account the probability of success in end-to-end transmissions analyzing the possible causes that lead to unsuccessful transmissions and delay considering the active neighbors that may cause contention on 802.11 MAC. The route quality metric is then the product of both effects through every link on a path.

The conflict graph is used to model co-channel interference, the channel assignment is performed as a coloring graph problem (colors represent channels) solved by a genetic algorithm and backtracking to reduce convergence time.

Each node that doesn´t have a route to a destination starts by broadcasting RREQ packets to its neighbors containing the RREQ ID, destination and source addresses, intermediate nodes, active neighbors, and a timestamp. All nodes rebroadcast this message until it reaches destination. Receiving a RREQ packet a gateway node, along with all other gateways interconnected on the wired infrastructure, collaborate with each other on channel allocation and route selection. The RREP is sent through the gateway and route that maximizes link quality. Intermediate nodes forward the RREP back to the source updating the routing table entries and performing channel switching if required. Elder entries are discarded to guarantee good quality routes as the network scenario varies.

2.3.5. LINK LAYER PROTOCOL (LLP)

In [28] the authors address the challenges of the utilization of the available channels on limited interfaces and propose a new architecture based on a source initiated on-demand routing protocol similar to Dynamic Source Routing (DSR). They also purposed a new

routing metric called Multi-Channel Routing (MCR) that extends the Weighted Comulative Expected Transmission Time (WCETT) metric to account for the inequality on the number of NICs and available channels so as switching costs. MCR metric considers routes over diversified channels while it takes also into account the cost of channel switching.

To address the problem of having more channels available than the number of NICs the LLP classifies the available interfaces as fixed and switchable. Fixed interfaces stay on fixed channels (FC) for long periods of time while switchable interfaces can be switched more frequently, as necessary among non-fixed channels giving higher priority to channels with oldest queued data. Switchable interfaces only change channel if there are packets to transmit on other channels and the current data queue is empty or a timeout condition is reached. Each node maintains two tables: A *NeighborTable* containing the fixed channels being used by its neighbors and the *ChannelUsageList* containing how many two-hop neighbors contain the same FC. When a packet reaches the link layer the fixed channel of the destination is looked up in the *NeighborTable* and the packet is added to the correspondent queue. To enable broadcast messages, a copy of the message is added to the queue of each channel on switchable interfaces. Figure 13 shows the association of queues to interfaces.

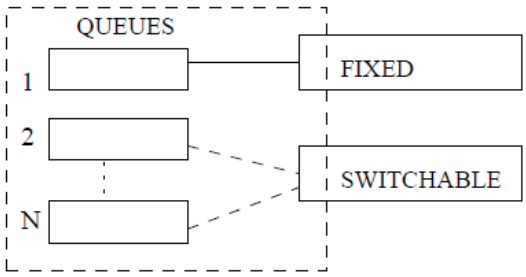


Figure 13 - Queues association with channels [28]

To keep each table up to date HELLO packets are exchanged between nodes periodically containing the node's fixed channel and the current *NeighborTable*. Receiving the HELLO packet a node will update its *NeighborTable* entry with the fixed channel of the sending node. Likewise the *ChannelUsageList* is also updated using the *NeighborTable* on the HELLO packet, which ensures a two-hop channel usage information. This table will be consulted before a node initiates HELLO transmissions and the fixed channel switched with probability *p* if the number of neighbors using the same fixed channel is high.

Route discovery process is initiated by a node broadcasting RREQ packets over all channels. This packet will contain the ETT, switching costs and channels used on all previous hops. Upon receiving the RREQ intermediate nodes will rebroadcast the packet if the sequence number is being seen for the first time or if the cost of the already discovered path is smaller than other seen on an earlier received RREQ. When the destination receives the RREQ will reply with a RREP packet only if the cost of the received RREQ is smaller than previously received packets with the same sequence number. The sequence number is maintained as intermediate nodes rebroadcast the packet. Additionally a procedure of route refresh is used ensuring that route cost is up to date and new routes with lower costs are discovered.

Routing in LLP is performed on layer 3 and is very similar to Ad-hoc On-Demand Distance Vector (AODV) with the exception that nodes don't store routing tables but instead source routing is used so packets must contain the entire path to destination introducing considerable overhead. This is also a protocol aimed at Ad-hoc networks so it doesn't consider that most of the traffic might be directed at a gateway placed on the wired infrastructure. Although it considers multicast traffic the methods used are not particularly efficient.

2.3.6. FIRST RANDOM CHANNEL ASSIGNMENT (FRCA)

The authors in [29] started first by studying the optimal number of radio interfaces on a node in a common channel assignment. Then the First Random Channel Assignment algorithm (FRCA) is proposed. It's a dynamic and centralized load-aware channel assignment and routing algorithm. Channels are assigned according to the expected load and the effects of inference of other links which are in interference range and tuned to the same channel.

It consists of two phases. In the first phase it estimates the loads in every link created by the routing algorithm and randomly assigns channels to the interfaces of each link. The procedures in second phase are similar to those used in the first phase. Links are visited in decreasing order of the link expected load. A link with higher load than the previous bandwidth/capacity provided will be revisited and assigned a channel that can fulfill the demand. This is an iterative algorithm that runs until the demands on all links have been fulfilled. To find routes between nodes FRCA uses shortest path routing metric based on hop count metric.

For FRCA authors concluded that the optimal number of interfaces is six evaluating its performance using between 2 to 8 interfaces on each node. Iterative algorithms like FRCA scale in complexity with network size. It is also a centralized solution where the global view and network state is needed to gather and disseminate information. As a load-aware algorithm it is based on the previously known traffic patterns. Routing layer is not specified.

2.3.7. MCUBE

MCUBE [30] is a modular MCMR WMN scheme with two major components. The Split Wireless Router (SWR) and the Channel Management Server (CMS).

The hardware architecture aims to reduce self-interference (due to nearfield effect, inter-radio board crosstalk and radiation leakage) is built from Commodity of the Shelf (COTS) devices with three physically separated processing nodes termed Radio Unit (RU) operating in orthogonal frequencies. Each RU can operate in heterogeneous technologies like Wi-Fi, WiMAX or Bluetooth, however the RUs are interconnected via an Ultra Wide Band network backhaul. The split router architecture, depicted in Figure 14, enables the construction of multi-radio routers equipped with radios of the same band with a physical separation of 0.5m and 40 MHz.

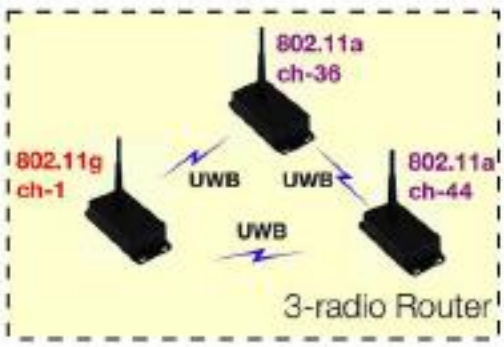


Figure 14 – MCUBE SWR [30]

To simplify the routing protocol and network management tools a *hardware abstraction layer*, shown in Figure 15, is important so each split router appears as a single-unit router equipped with multiple interfaces. Among the three RU of the SWR one (that operates on a physical layer common to all SWR on the mesh) will be denoted as the Designated Radio Unit (DRU) where software such as the routing protocol and network management tools are

hosted exclusively. Each RU will broadcast its identity (at every minute) and also the observed neighbor set among them, over the backhaul network.

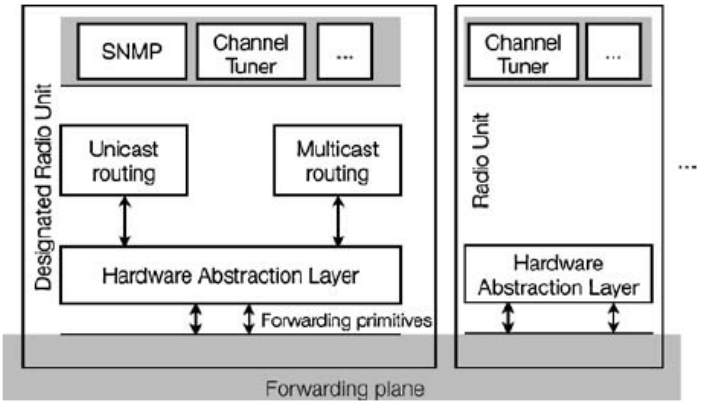


Figure 15 – DRU internal Architecture [30]

Channel selection in MCUBE architecture is performed using the proposed Topology and Interference-aware Channel selection algorithm (TIC). To perform TICs operation the network topology needs to be discovered for each technology supported by the mesh network (topologies vary according to frequency bands and channels).

While performing topology discovery a single-radio mesh is created over a common channel orthogonal to the one used for topology discovery in order to keep active flows. The creation of the SR mesh is issued by the CMS with the broadcast of a Default-switch message which contains the channel number. Once the single-radio mesh is created CMS notifies all the other radios to tune to a specific channel. While performing network discovery each router also measures link quality to its neighbors using the expected transmission time (ETT), packet pair probing to estimate the bandwidth and expected transmission count (ETX) for loss rate. The CMS runs TIC.

The channel assignment information issued by the CMS is spread over the network using unicast CHANNEL-INFO messages through the default mesh. The CMS then broadcasts a CHANNEL-SWITCH message that will be rebroadcasted by each node radios and then the cached routes will be deleted. The new routes will be afterwards assigned. Any routes required for packet delivery after the switch completes are discovered using reactive route discovery (SRCR).

MCUBE modular design enables the capacity to expand and adapt the Mesh network and to support multiple technologies. With the proposed 0.5m of NIC separation module dimensions can be a drawback in some scenarios. The topology control performed by TIC's algorithm might take too long have an impact and respond in short-time to topology variations. It is a centralized solution so inherit delays cannot be disregarded as well as the overhead from layer 3 routing.

2.3.8. GENERALIZED PARTITIONED MESH NETWORK TRAFFIC AND INTERFERENCE AWARE CHANNEL ASSIGNMENT (G-PAMELA)

G-PaMeLA [31] is a divide-and-conquer technique which divides the overall Joint Channel Assignment and Routing problem into a number of sub-problems equal to the maximum number of hops to the gateway. Each sub-problem is formulated as an Integer and Linear Programming (ILP) optimization problem. It is a centralized solution where a dedicated NIC is used for control purposes. Periodically the gateway runs G-Pamela to assign channels and to determine routing paths being also responsible to disseminate the updated channel assignment and routing to all nodes.

The algorithm starts first by gathering information about the physical topology as a graph *G(V,E)* where *V* is the set of nodes and *E* unidirectional links, the number of nodes, NICS on each node, the power of the received signal, channel rates and traffic load. Once it has all information advances to the JCAR phase where the main JCAR problem is partitioned into a set of ILP sub-problems. This division is based ranking functions considering network topology and load (distance to gateway and traffic load); nodes closer to the gateway have highest rank since they relay more traffic. Although different ranking functions can also be used. Because there is no guarantee of the existence of logical links after the JCAR phase a second phase for each resulting solution is applied as post processing phase which ensures connectivity along the network and that no NIC is unused. The final result is an optimized set of routes and channels to be used in the WMN that is disseminated through the network.

As a centralized solution G-PaMeLa infers delay and overhead. It may not respond well to fast changes on the network because of the amount of information it has to consider and compute to provide an efficient solution. Also, as before mentioned the use of a dedicated control NIC introduces a waste of resources.

2.3.9. MULTIPLE ACCESS SCHEDULING IN MULTI-RADIO MULTI-CHANNEL MESH NETWORKS (M4)

M4 [32] is cluster based solution that uses Latin Squares. Three different ranges are considered Transmission Range (Rtx), Carrier Sensing Range (Rcs) and Interference Range (Ri). The collision domain is defined as the set of adjacent nodes that can interfere and damage packet transmission. Nodes within each collision domain are fully connected with each other in the two-hop range and are clustered to be a clique in the network graph. Nodes within collision domain share the same channel for access scheduling and are aware of each other.

M4 is mainly composed by four schemes: the Interference and Bridge clustering, the on-demand coloring based multiple access scheduling and the MRMC scheduling oriented routing.

Nodes within each collision domain are fully connected in the two-hop range and form a clique in the interference graph. Assuming that in a large WMN nodes are grouped within non-overlapping collision domains each collision domain forms a clique in the network graph.

The WMN is structured in two types of clusters: Interference Clusters (IC) and Bridge Clusters (BC). Nodes within the first cluster form collision domain of each other and use the same channel. Nodes on the BC also form the collision domain of each other except that they are chosen among adjacent IC to facilitate the communication between clusters that use different channels. To build the IC the node with the highest degree, named seed, builds its one-hop neighborhood. After seed neighbors are visited and are fully connected the clique is formed, removed from the network graph and added to the network clique cover. The next clique can then be created. It's possible to limit the size of each clique so as to reduce the channel resource contentions in each clique. This process is depicted in Figure 16.

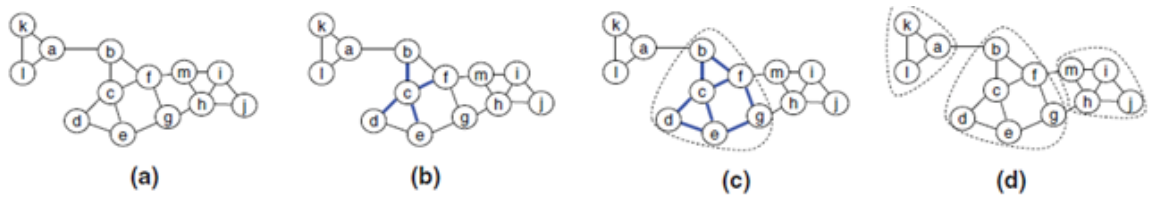


Figure 16 – Cluster formation in M4 [32]

The Bridge Cluster, Figure 17, is created by removing all links within each interference range and the subsequent isolated nodes resulting in intra-cluster connections only. Then the node with the most one-hop neighbors is selected to be the seed node. The rest of the procedure is the same as creating an IC.

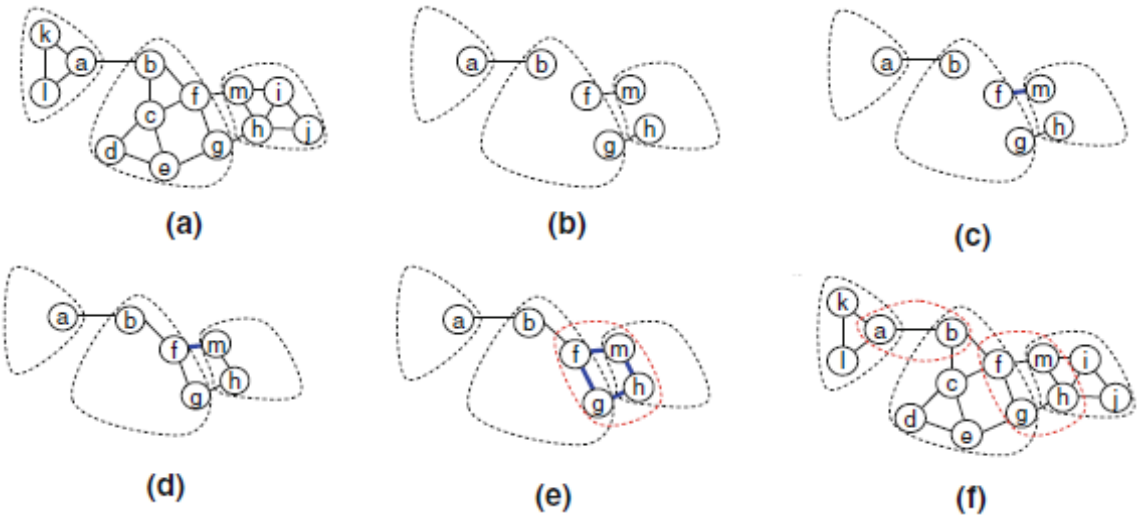


Figure 17 – Bridge Cluster formation in M4 [32]

Two types of graph coloring schemes are used for the network, node coloring and cluster coloring. To achieve the conflict free within interference or bridge clusters color assignments the coloring information of each node is sent included in the header M4 data frames (*nodeid, mycolor, onehop*). Once the color assignment is completed the Latin Square row assignments to different node is achieved by mapping the color information to the Latin square row indices. Cluster coloring prevents interference from close range interference clusters with the same channel assigned, by replacing the node entity by the cluster entity. First channels are assigned to clusters.

M4 route establishment is achieved by RREQ and RREP messages. The optimal route is defined by the authors that proposed a routing metric called *forwarding speed* as the route with less forwarding hops, higher data rates and lower queuing delays. Because the number of neighboring clusters of a node can be more than the number of available interfaces that can cause potential delays over the network, the second highest rated node is selected as the alternative.

Uses the number of radio interfaces is not defined, although it used a dedicated control NIC. It is a distributed coordination scheme, in which performance was evaluated on simulation environment.

2.3.10. ROBUST JOINT CHANNEL ASSIGNMENT AND ROUTING WITH TIME PARTITIONING (RCART)

RCART [33] is a protocol that considers the cyclic behavior in network traffic and uses this knowledge to build its routes and assign channels to interfaces. Traffic is considered to have regular periodic trends over large periods of time and also to be spatially variable i.e., different APs may have different demands over different periods of time.

Routing is performed considering the optimal routing solution, solved as an LP problem, to derive the routing solution that minimizes wireless network congestion for a given set of demands. It is divided into three distinct phases. In the first phase named *Time Partitioning and Traffic Characterization* time is partitioned into periodic intervals with consistent properties. In the second phase the objective is to find a routing scheme that provides the upper bound to the worst case of traffic scenario following two major constrains: to ensure channel assignment and scheduling by guaranteeing that a channel is not oversaturated and that nodes do not have more channel time assigned to them than radios. Finally channel assignment is performed binding radios to channels based on the traffic distribution resulting on channel diversified paths with minimal congestions and collisions.

Channel assignment is performed in a distributed with nodes initially mapping all flows to a SC, then to each flow the assignment that maximizes the capacity is computed. As other channels are used the capacity will increase leading to a local optimum. Further on, a flow is randomly chosen to test reallocation and if it improves capacity then it's performed.

Traffic between the mesh APs and the internet is assumed to be infinitesimally divided and routed over multiple gateways to achieve minimum congestion and maximum balance, which in practice isn't always true. Also, as a traffic history based approach RCART relies on a great number of samples taken over a large period of time to optimize the network profile because some periodical trends can only be observed on a large time scale. The distributed channel assignment scheme may take several time to converge to an optimal solution and introduce high contention even on heavy flows.

2.3.11. JOINT MULTI-RATE (JMR)

Authors in [34] consider not only the joint optimization of routing and channel assignment but also the interdependency between these two and link-rate allocation (LRA) and propose an heuristic algorithm. LRA enables that different links use different rates (not only the highest rate) improving this way network capacity through spatial reuse. They also propose a new architecture, shown in Figure 18, to reduce the number of gateways with specific nodes placed around the gateway called *ringnodes* connected to it with high speed connections that do not interfere with the regular WMN transmissions.

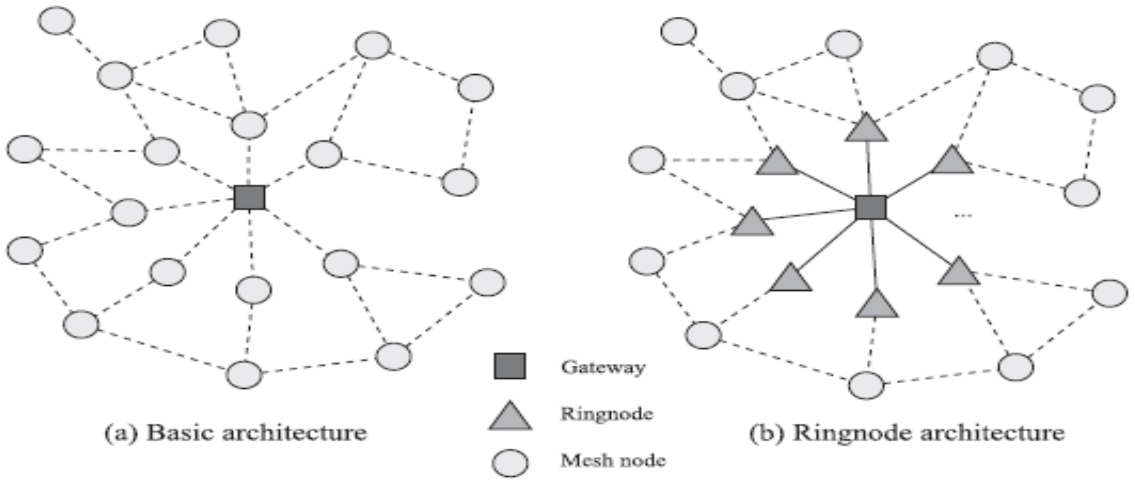


Figure 18 - Architectures used in JMR [34]

Each pair of nodes only communicate over one channel and the number of channels allocated to a node does not exceed the number of radio interfaces. To model interference they defined a Multi-Rate Interference Map (MrIM) that takes as input links transmitting on the same channel and at the same time with different rates and the Signal to Interference Ratio (SINR) of each one.

The algorithm starts by employing heuristic to find a good balance on link rates. As the space of LRA combinations might be too large unnecessary edges are removed from the connectivity graph *G* to speed up the heuristic method. Negligible links are the ones that do represent low quality and redundant links between nodes and the gateway. The Search Interference Range Space (SIRS) heuristic method is used then to find the good LRA parameter for each link with which routes are calculated and channels assigned. It takes as input the desired interfering range (as a function of the rate) to calculate the highest transmission rate for each link. Each solution is then evaluated constructing the conflict

graph for a fixed LRA and jointly solving the routing and channel assignment problem to maximize capacity as a Mixed Integer Linear Programming.

2.3.12. DISCUSSION

ROMA, Hyacinth, LLC, M4 and JMR implement distributed coordination. Centralized approaches like MCUBE, LACA, FRCA, G-PAMELA and RCART require gathering information from all network before taking any actions. Such operation increases in complexity as the network grows because longer intervals are required to gather information so as to compute it. Although in small scale networks they should perform generally better than most distributed solutions.

JRCA, MCUBE, G-PAMELA and M4 use a control dedicated NIC. Control messages aren't sent very frequently in any of the proposals, so this NIC will be idle most of the time. For the same reasons, some solution also misuse the available spectrum by allotting a control dedicated channel, like MCUBE. ROMA also proposes nodes equipped with only two radios (although not strictly).

Hierarchical topologies, like the one used in ROMA, may lead to unfairness since high quality links are assigned to nodes closer to the gateway. Although, the aggregated traffic increases as links are closer the gateway which may benefit the network's overall performance. M4 clustering scheme introduces another level of computation and delays, as channels are assigned to clusters, and then to nodes on each cluster. This type of approach suits well in large scale networks but is inefficient otherwise.

3. IEEE 802.11 WIRELESS PROTOCOL

An introduction to some theoretical concepts is provided in this chapter with the objective of giving the reader the necessary background to understand the key topics covered in this dissertation. Our main focus are Infrastructured networks in a typical scenario of Access Point (AP) – Station (STA).

3.1. PROTOCOL ARCHITECTURE

The IEEE 802.11 building block is the Basic Service Set (BSS). It comprises the set of terminals that communicate with each other over the same channel controlled by the Coordination Function (CF). Two basic types are defined: the Independent Basic Service Set (IBSS) and infrastructure Basic Service Set. In IBSS scenarios each STA creates its own service set and may communicate directly with other STAs within communication range.

In IBSS mode, also named ad-hoc, Figure 19 each station creates its own service set and may communicate directly with stations is range. The main focus of such an implementation is often a scenario where nodes have the need to communicate through short periods of time, due for example to mobility. The coordination is distributed among the peers and no node plays a master role.

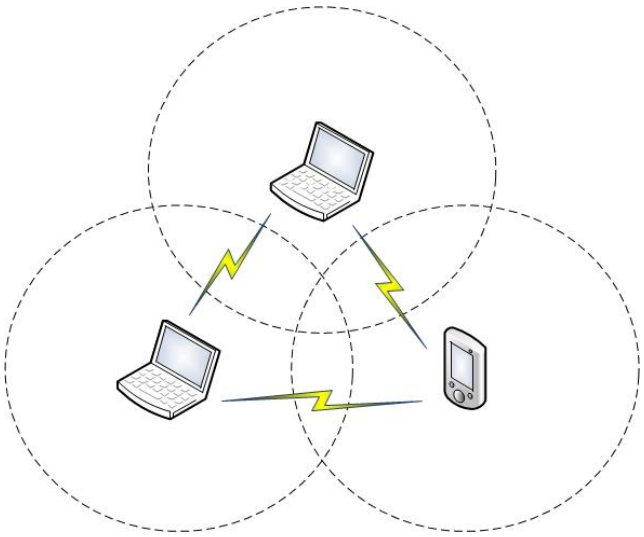


Figure 19 – Simple Ad-hoc network

Infrastructured deployments, Figure 20, are distinguished by the use of an AP that coordinates and forwards all communications that occur in that BSS including those between nodes within the same BSS. Additionally when multiple BSSs are interconnected through a distribution system the term Extended Basic Service Set (EBSS) is often employed.

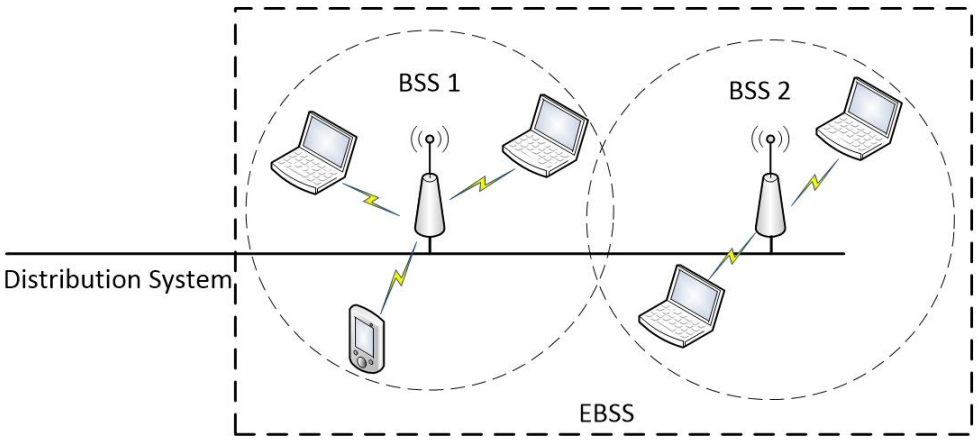


Figure 20 – BSS and EBSS

3.2. MANAGEMENT FRAMES

In order to establish a basic set of services in a wireless network, so as to provide connectivity, management frames are issued among participant stations. These frames are mostly constituted by variable-length blocks of information denoted Information Elements (IE). The generic structure of an IE is represented in Figure 21.

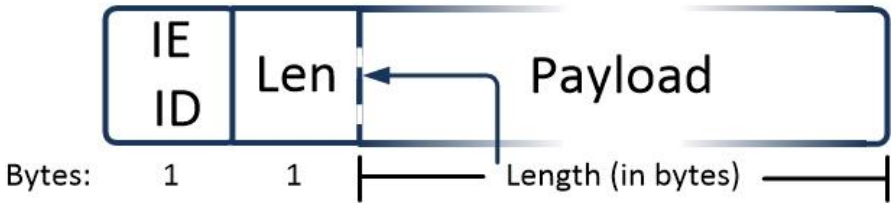


Figure 21 – Generic IE structure

The ID represents the standardized value of the IE, while the Length represents the size in bytes of its payload. Some of these management frames are described in more detail below.

3.2.1. BEACONS

Beacons are transmitted by IEEE 802.11 APs in order to announce the existence of a network, in fact they define the BSS area and provide external stations the key parameters that must be met to joint it. Also they have an important role in network maintenance. Within a BSS they may be used to define contention free intervals, and most of all, to keep station synchronized, so as many other functions of network maintenance defined in the different IEEE 802.11 standards. Their periodicity is not tied to a specific value although a default rate of 10 beacons per second is set (100 ms). Not all IE are mandatory as some may only be available in beacons when needed. Figure 22 shows a beacon frame.

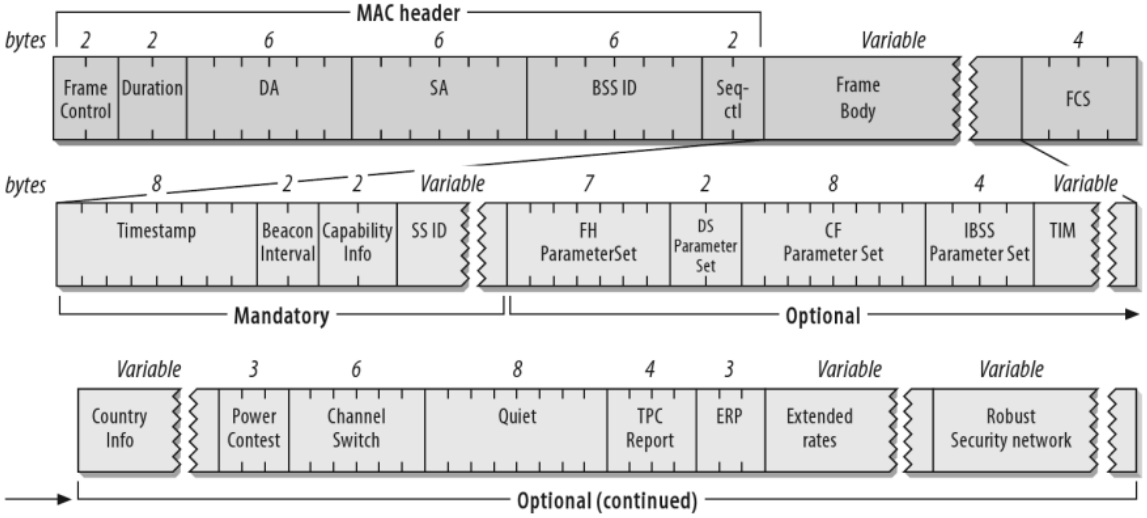


Figure 22 – Beacon frame [35]

Beacons are used in this dissertation context to both advertise new MAPs about the existence of a network and provide essential parameters to efficiently extend the wired infrastructure.

3.2.2. PROBE REQUEST/RESPONSE

Stations with no association query their surroundings using Probe Request frames. These are small sized frames typically containing the essential information needed by a station before association, like supported rates and SSID. Although SSID field could be set to the broadcast address if a station pretends to join any compatible BSS. When an AP listens to such information it might decide to accept the new station or reject for whatever reason. Compatible supported rates and SSID are fundamental matching parameters. If the station is free to associate, the AP then sends a unicast Probe Response frame, which structure is similar to the beacon, containing all information needed to the station to match parameters and join the network.

3.2.3. ASSOCIATION/DISASSOCIATION AND AUTHENTICATION/DEAUTHENTICATION

Authentication/Deauthentication and Association/Disassociation frames establish/break relations at two different levels respectively. The main objective here is to validate the device using either a two way handshake (open system) or a four way handshake (shared key). Authentication occurs prior to Association. Here, after receiving the request the AP will send the reply frame containing a success message, if it is an open system, or the authentication algorithm and challenge text if shared key is used. The association pairs request/reply complete the four-way handshake. Not much detail will be given here, since security issues fall out of scope of this dissertation.

Disassociation and Deauthentication frames are used to break the respective level of commitment to the cell. Both frames are very similar and both include a mandatory reason field.

In the proposed solution connections are established using a two-way handshake. Authentication and Deauthentication will be used to keep routing tables up to date.

3.2.4. CONNECTION ESTABLISHMENT

Putting all afore mentioned frames together, the connection establishment in an infrastructured deployment might be described as follows.

Stations typically perform two types of scans: passive and active scans. When passive scans are performed a station stands quiet and listens to beacon frames broadcasted by neighbor

APs. Alternatively, a station may start immediately querying the neighborhood, using probe request frames for available BSSs. When the method used is the former it is called active scan. The default behavior it is listen then query.

After issuing all probes the authentication and association processes follow. While associated to the AP the STA will compete with other stations within the BSS for a slot to communicate. All STA sync their clocks using beacons sent from the AP. When the AP decides it may decide to exclude an STA from the IBSS, for example due to inactive time sending a Deauthentication frame.

3.3. LINUX WIRELESS MANAGEMENT ARCHITECTURE

The Linux Wireless Subsystem is divided into two major blocks: the data system and the management system, which can be seen as mostly independent from each other. We will just highlight the management path since, it is the one that our proposed solution is mostly concerned. The management system handles all the processes described in Section 3.2. The basic architecture is depicted in Figure 23.

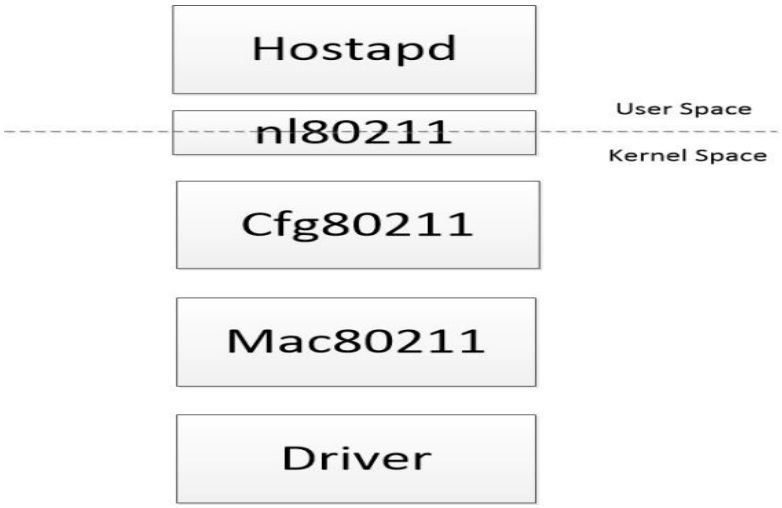


Figure 23 – Linux Wireless Management Architecture

3.3.1. HOSTAPD

Since the implementation of SoftMAC devices in Linux (devices that do not implement the MAC layer in hardware, as opposed to Hard/FullMAC) hostapd is the preferred way of configuring an interface to Master (AP) mode. This daemon includes IEEE 802.11 access point management, IEEE 802.1X/WPA/WPA2 Authenticator, EAP server and radius

authentication server functionality. Accordingly *hostapd* will generate template of IEEE 802.11 control messages and handle some of management tasks of the MLME layer, like authentication and association requests, leaving other protocol critical operation to be handled in kernel space, like sending beacons which are time critical.

The access point configuration is loaded from a customizable file where the minimum options (interface, driver, SSID and channel) should be present to enable the AP functionality. While running, some configurations may also be changed using the control interface, which enables Inter-Process Communication (IPC) support using a UNIX domain socket, through which external programs can either control or receive status and event notifications. Commands are used to perform control and consist in pairs of request/reply from the external program and *hostapd* respectively. Event notifications are unidirectional and the external programs only need to subscribe to the socket and start listening to the event directed to user space without explicit request.

3.3.2. NETLINK (NL80211)

Netlink sockets are an IPC mechanism used for data transfers between kernel modules and user space applications. At user space the standard socket Application Programming Interface (API) is used and a special kernel space API for kernel modules, both providing asynchronous full-duplex communication links.

Each *netlink* communication channel is associated with a family which in turn associated with a service. Examples are the *NETLINK_ROUTE* family which focus mostly with routing tables, enabling interaction with routing tables and control of other parameters IP addresses and traffic classes [36] and *NETLINK_FIREWALL* and *NETLINK_IP6_FW* that respectively enable transport of IPv4 and IPv6 packets to user space [37]. The most recent family is the Generic family, created with the concern of the exhaust of the number of families which was till then limited to 32. The Generic *netlink* family behaves as a *netlink* multiplexer allowing other services to register to this family. The wireless subsystem is one such service that registers itself with the name nl80211 and gives to user space applications like *hostapd* a communication channel to the kernel.

Since *generic netlink* messages are used in context of the proposed solution, these will be given more focus. The format of the *generic netlink* message is depicted in Figure 24.

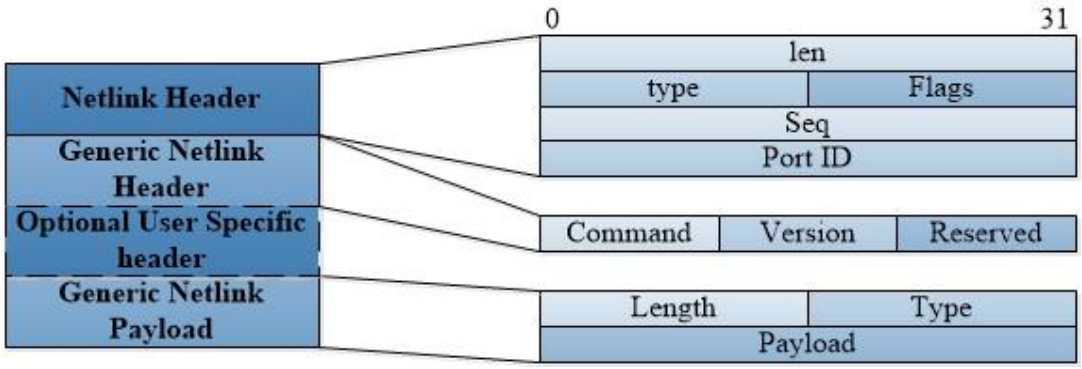


Figure 24 – Netlink *generic* message format

Netlink header carries general information about its contents. The *length* field specifies the total length of the message. *Type* and *flags* respectively identify the content and guide the recipient to process the message. The sequence number identifies the message, although the *netlink* protocol doesn't apply any strict enforcement about it. The port ID either specifies the port or the process id if the source of the message is the kernel (port 0) or a process from user space.

The *generic netlink* header specifies a *command*, which is specific for any family registered under the Generic family, *version* if any version exists and 2 bytes reserved for future use.

The payload field in *netlink* messages, either concerning generic or any other family, assumes a recommended format similar to the one used in IEs, with attributes represented on the format *type*, *length* and *value* which must be align to a four byte boundary.

3.3.3. CFG80211

Cfg80211 is the configuration API for Linux 802.11 and can be seen as the kernel interface for user space applications, providing also essential management services for both full and soft mac devices.

Structures used by kernel space modules are registered using cfg80211, these comprise for example the *wiphy* structure that includes the hardware capabilities and beacon parameters with pointers to static parts of the beacon.

3.3.4. MAC80211

MAC80211 defines a softMAC device. Since placed in the middle of the Linux Wireless subsystem it implements both callbacks to interact with cfg80211for management and

provides a driver API which handles most of the management frame construction and parsing [38].

3.3.5. DRIVERS

Driver, at the lowest level in the kernel, provide the API that enables other upper system to handle the hardware seamlessly.

3.4. IEEE 802.11S

IEEE 802.11s is an extension to the IEEE 802.11 standard that defines a tree-based routing solution, based on Layer 2.5, to extend the wired infrastructure. Routing is performed using the Hybrid Wireless Protocol which defines two mechanisms for topology establishment, one is the proactive Path Request (PREQ) mechanism and other is the proactive Root Announcement (RANN) mechanism.

The proactive PREQs are issued from the GW to announce its presence, and MAPs use these message to create and update the path to the root. These metrics propagate all the way through the network. RANN messages are also issued proactively from the root node, but have only the purpose to update metrics at each node, instead of also update routing path. This aims at proving nodes, the necessary to establish the best paths to the GW.

4. PROPOSED SOLUTION

We now present a novel distributed multi-radio multi-channel solution to wireless multi-hop networks. As seen in the state of the art section, one of mesh networks main weakness is the waste of resources. In a struggle to improve efficiency most stated solutions rely on dedicated control messages, radios and channels. Our proposed solution is focused both on reducing control overhead while using the minimum number of radios without changing IEEE 802.11 standards. We also break the common scenario, of a mesh formed by multiple ad-hoc links, and use multiple BSSs to create a multi-hop network.

As afore stated, the IEEE 802.11 protocol already incorporates mechanisms to periodically broadcast information within an area. Specifically, APs use beacons to announce their presence and inform other stations about which configurations should be used to connect to that BSS. These are, in fact, the same characteristics needed in control messages on mesh networks to disseminate metrics and improve the network overall performance. Thus, instead of using additional control messages we overload the IEEE 802.11 beacons, with a technique called beacon stuffing, to broadcast metrics and provide service announcement eliminating consequently unnecessary overhead.

Our proposed solution extends the WiFIX protocol to support multiple radios and efficiently use the available spectrum by operating radios in multiple non-overlapping channels over

both IEEE 802.11 b/g and IEEE 802.11 a. The architecture comprises fixed nodes each equipped with two interfaces organized in a hierarchical tree structure, with parents and their children as seen in Figure 25. At the root of the tree, a gateway node with only one interface routes traffic between the wired infrastructure and the WMN network. Each MAP in the mesh tree operates both as a station within a parent infrastructure and as the AP of its own BSS.

Nodes join the mesh listening to control information, disseminated embedded in beacon frames and always connecting to the parent that guarantees the shortest connectivity (in hop count) to the wired infrastructure. Our channel assignment strategy aims at the reduction of interference within the mesh, alternating between 2.4 and 5 GHz bands, at each hop, to take the most benefit out of the channels provided by IEEE 802.11 standard. Based on WiFIX, routing is performed on layer 2 using MAC addresses instead of IP addresses turning it a layer 2.5 solution. Each packet is decapsulated at each hop and sent over Eo11 tunnels.

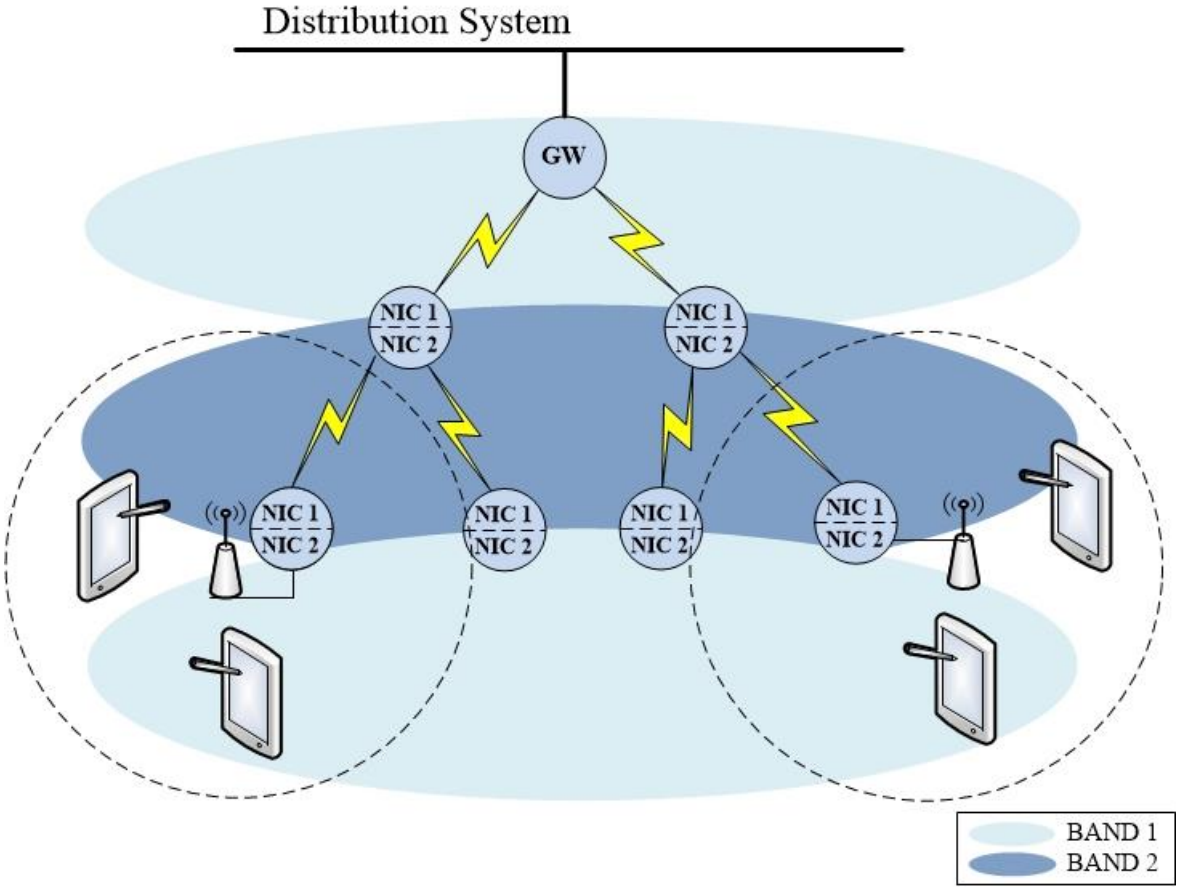


Figure 25 – Proposed architecture

MAPs are equipped with two interfaces each tuned to a different frequency band (2.4 and 5 GHz) and assign one to communicate with its parent and other to its children, in STA and AP mode respectively. As authors state [24], even when operating on orthogonal channels on the same frequency band, interference occurs if antennas are not separated by at least 45 cm. Using two radios on distinctive bands we ensure that no interference occurs due to close range operation while using small sized nodes. Also this will ensure that each collision domain is broken at each hop, no matter the criteria used for choosing channels.

4.1. OPENWRT

OpenWRT is a free, open-source and community driven Gnu/Linux distribution for embedded devices, specifically designed for networking purposes. It features a package management system making it highly customizable and easy to port custom software into, without having to build the whole system. Nodes run on stable releases 14.07 and 15.05, if equipped with Alix boards or TP-Link respectively.

Embedded systems typically use different processor architectures and require a compilation toolchain, at a host system, to generate code for them (target system). The compilation toolchain comprises a set of tools including: a compiler, binary utils like assembler and linker and a tiny C standard library, in this case *uClibc*. In this dissertation, the host runs a Linux, Ubuntu (Debian based) kernel 4.1 distribution over a x86 64 bits processor and ALIX and TP-LINK targets run on Linux 3.10 kernel (OpenWrt Barrier Breaker 14.07 RC3) x86 32 bits and Linux 3.18.17 kernel (OpenWrt Caos Calmer 15.05 RC3) and Microprocessor Without Interlocked Pipeline Stages (MIPS) processor, respectively. Each OpenWrt distribution includes a Software Development Kit (SDK) that is a precompiled toolchain apposite to cross compile single user space packages for the given target. The SDKs used were "OpenWrt-SDK-x86-for-linux-x86_64-gcc-4.8-linaro_uClibc-0.9.33.2" available at [39] for the x86 target and "OpenWrt-SDK-15.05-ar71xx-generic_gcc-4.8-linaro_uClibc-0.9.33.2.Linux-x86_64" available at [40] for the MIPS target. The pre-built images were used, since they fill all requirements in scope of this dissertation and are also available at [39] and [40] for x86 and MIPS. Appendix B is the developed *Makefile* to cross-compile the application.

4.1.1. CONFIGURATION FILES

System's key configurations, in OpenWrt, are split into a set of files located in */etc/config/* directory each related to a specific part of the system and manageable with a text editor or with Unified Configuration Interface (UCI), a configuration interface available that centralizes configuration of the most important system settings. The most important changes were made to network and wireless configuration files.

The network file defines all network related configurations like switch VLANs, interface configuration and network routes. The network configuration file is shown in Appendix C.

Wireless configurations handles the wireless subsystem in OpenWrt. The configuration of each radio is handled by the pairs *Wifi-Device* and *Wifi-Interface*. *Wifi-Device* refers to the physical radio interface and specifies mostly level 1 information like channel, txpower, protocol and wireless networks. The *Wifi-Interface* complements the radio interface parameters with the wireless network configurations, mostly layers 2 and 3 of the OSI protocol stack. The *wireless* configuration file is depicted in Appendix C.

4.2. INITIALIZATION

As in WiFIX, a single software portraits both GW and MAP capabilities, so before initialization the user is inquired to specify which node type he intends to run (GW or MAP), together with debug level and mesh interfaces name. Interfaces are kept in a structure array, where each structure contains the set of relevant parameters for interfaces, such as interface name, MAC address, supported wireless protocols and a socket file descriptor. Given the name, specified by the user for each interface, two different system calls to *ioctl* fill the MAC address and supported protocol on the respective interface structure with the information provided by the kernel. Each system call is performed inside *getMacAddress* and *getWirelessProto* functions, which code is depicted in Appendix E using two distinct preprocessor macros: *SIOCGIFHWADDR* to get the MAC and *SIOCGIWNAME* to get the supported protocols from *WEXT*. The specification of interfaces for MAPs follows the constraint that each should run on a different band.

On initialization the GW's interface is set to AP mode since no further alteration during run time will happen. The same doesn't happen to MAPs since the mapping of interfaces to

mode and protocol isn't known prior to parent discovery, so both are initially set to STA mode.

4.3. CONNECTION ESTABLISHMENT AND TOPOLOGY FORMATION

Connected to wired infrastructure each gateway is the root of a spanning tree that each MAP will try to participate in. Before association each MAP will passively scan the environment by listening to beacons and reading the vendor specific IE. Initially, only gateway beacons will be available and these can be identified by hop 0 and no channel usage information.

4.3.1. CONTROL MESSAGES

In protocols that implement distributed coordination each node must make careful decisions whether if it is assigning channels, routing traffic or simply joining the network, as the smallest decision may have a huge impact on network performance. With an interference range twice the perceived/carrier sensing range a node can't simply rely on its perceived environment to make channel assignment decisions. Hence, control messages propagate through the network to give nodes input metrics as guidance in decision-making. This messages may either be sent periodically or in an on-demand fashion so as broadcasted or unicasted.

Our control information is disseminated embedded in beacon frames. More specifically, we overload the IE 221 to broadcast information about network topology (number of hops to the gateway) and channel usage throughout the network. Each node will broadcast this information enabling network expansion without disregarding channel interference with up-neighboring nodes.

4.3.2. BEACON STUFFING

Beacon Stuffing [41] is the name given to the mechanism that enables broadcasting non-standard information in beacon frames. Four different techniques may be utilized to carry custom information in beacons, each differentiating in the field that is used to carry the information.

The SSID is one such field. With a maximum length of 32 bytes is enough to carry our network metrics and its modification is feasible from user level. One disadvantage of overloading the SSID field is the increasing in length as the metric increases. It also limits

the implementation to scenarios where SSID is not hidden and some operating systems may not permit changes in this field without rebooting the access point what would result in a cascading effect of disassociations.

The six byte length Basic Service Set Identifier (BSSID) field also provides room for information embedding. This field is usually set to be equal to the AP MAC address although it can be set to any value. With only six bytes, it imposes high restrictions on the metric.

Some authors [42] also suggest the use of length fields of the information elements (IE), more specifically the unused bits in length fields. In fact most IEs don't use some of the most significant bits available in length fields and with a limit imposed on the maximum size by the IEEE 802.11 standard extra information (about 191 bits) may be added. This procedure lets us take full advantage of the available resources on IEEE 802.11 networks. The drawback is the complexity of such implementation. If a variable length metric is used the mapping of such parameters in length fields is highly complex and may not always be feasible.

The vendor specific IE is an IEEE 802.11 standard and allows each vendor to add specific information elements to beacon frames. Each of these elements is limited to 252 bytes of payload (256 bytes in total minus 3 Organizationally Unique Identifier (OUI) bytes minus 1 byte to specify the type). Additionally more vendor IEs may add as long as the beacon frame body does not exceed 2320 octets. Because overloading the vendor elements is the most flexible solution we used to disseminate metrics and scale our network. Also, it is possible to alter the information contained in the IE without breaking connection with associated stations. The general format of the vendor information element is depicted in Figure 26.

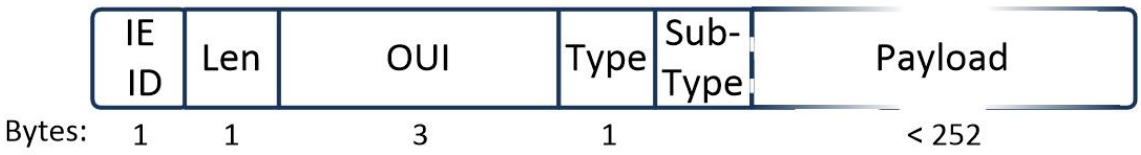


Figure 26 – Vendor Specific Information Element

The vendor element is identified by the element ID 221 and it is always the last IE in a beacon. The length field follows, as in other IEs it sets the size of the information carried (this includes OUI, type, sub-type and payload). The OUI field contains the globally unique identifier assigned to each organization by the IEEE Registration Authority. To test our

solution we adopted an OUI that was not yet assigned to any organization [43], FF-FE-00. The type and sub-type fields are extensions to the OUI field being mostly used for Wireless Multimedia Extensions (WME) [44] and WPA improvements. We set the type field to a constant value of one while the subtype is overwritten with the topology information (hops). The payload field contains the actual data carried by beacons where we place an ordered list of channels used by upstream neighbours.

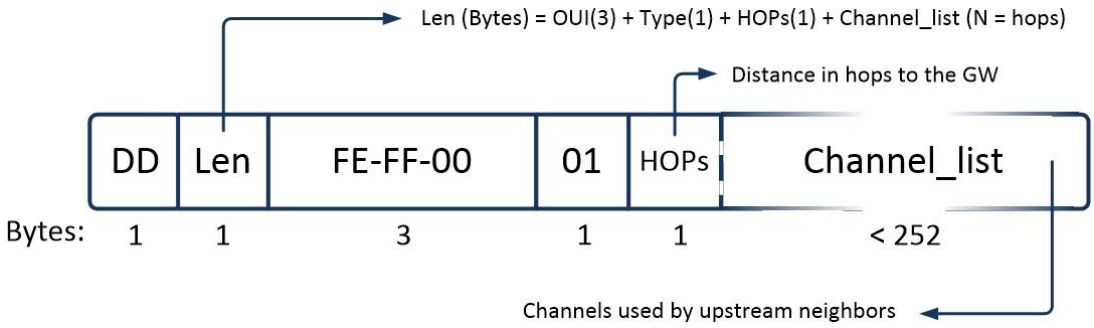


Figure 27 – WiFIX Information Element

One of the used methodologies to broadcast beacons was to develop a script that would be running in parallel with the main algorithm and injecting the modified beacons into the network. To avoid competing with authentic beacons, the beacon interval from the interface set to AP could be set to a much higher value.

Two different tools were explored to perform packet injection into the network. PyLorcon2 is a python wrapper for the LORCON2 C library used to inject 802.11 frames. The developed script using *pyLorcon2* is presented in Appendix F. The fact that this module had no recent updates and its maintenance seems somehow divided [45] [46], lead to also explore *Scapy*. *Scapy* is both a python package and a program, that enables packet injection, manipulation and sniffing. In fact *Scapy* provides much simpler mechanisms, compared to *pyLorcon2*, to inject packets, regarding the intended purpose. The script developed using *Scapy* is depicted in Code Excerpt .

```
#!/usr/bin/python

import sys
import random
from scapy.all import *

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print "Usage"
        print "  %s  <iface> <info>" % sys.argv[0]
        sys.exit()
```

```
    broadcast = "ff:ff:ff:ff:ff:ff"
    bssid = "01:01:02:02:03:03"
    interface = sys.argv[1]
    ssid = "WiFIX MRMC"
    #Generate a random sequence number in interval 0-4096
    seqcontrol = random.randint(0,4096)
    beacon_info = sys.argv[2].decode("hex")
    #beacon_interval = 0.2 #seg

    # Create the beacon structure
    beacon = RadioTap() / Dot11(addr1 = broadcast, addr2 = bssid, addr3 =
bssid, SC = seqcontrol) / Dot11Beacon(cap = 0x1104) / Dot11Elt(ID=0,
info= ssid) / Dot11Elt(ID = 1, info="\x80\x84\x8b\x96\x0c\x18\x30\x48") /
Dot11Elt(ID = 3, info = "\x04" ) / Dot11Elt(ID = 5,info =
"\x01\x02\x00\x00") / Dot11Elt(ID = 221, info = beacon_info )

    sendp(beacon, iface = sys.argv[1], count = 100, inter = .1)
```

Code Excerpt 1 – Beacon Injection script using scapy

In the above script, user inputs the interface to where beacons will be directed and the payload of the IE, including OUI, type and data. The beacon is then constructed from head to tail. The radiotap header is included in framed while capturing traffic in monitor mode, and gives picture of radio conditions at time of reception. Since the frame is to be sent no information is added. The Dot11() field appends the MAC header elements to the frame, including destination address, BSSID, source address and a sequence number randomly generated. The additional AP capabilities precedes the information elements, calling *Dot11Beacon* to specify this parameter. Each IE is identified by its ID and value. Although this script only contains the mandatory fields and the *vendor_specific*, all the other IEEE 802.11 standard IEs can be included following the same procedure. In fact, the injected beacons should have the same structure as the original beacons. Finally, one hundred beacons are sent with a periodicity of 100ms. Figure 28 shows a capture of the injected beacons.

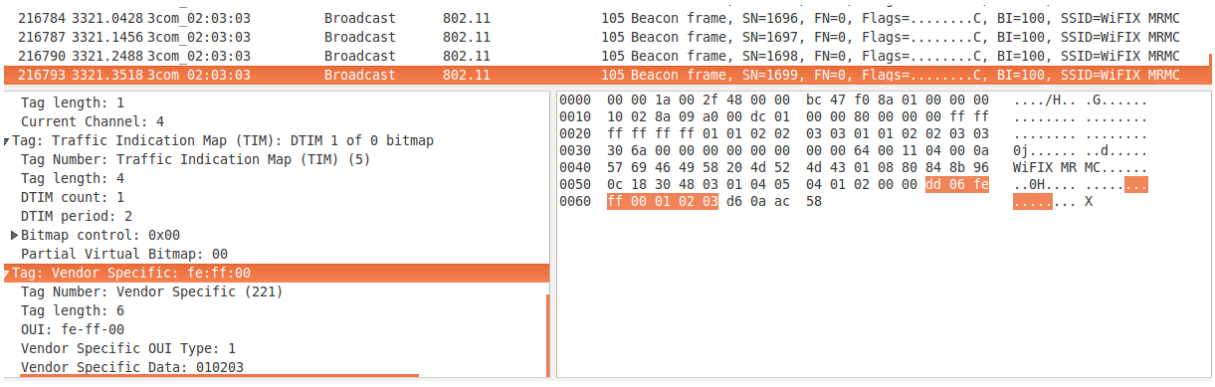


Figure 28 – Injected Beacons Capture

Packet injection scripts provide easier mechanisms to broadcast beacons with control information embedded in the information elements since the entire packet in built from user space without complex procedures. However, this approach breaks with the main focus of this dissertation. The injected packets should be sent over an interface running in monitor mode, which requires either an additional interface dedicated to control messages or a virtual interface. The injected beacons are additional packets sent with control purposes and lead to the same overhead resultant from other control messages. Also, uncertainty arouses concerning the state machine of the associated stations, since the sequence number of injected and original beacons would be different. As stated, the sequence number of the injected beacons is randomly generated by the script. Original beacon's sequence number is filled by mac80211 (*net/mac80211/tx.c*) and modifiable by the driver in kernel space (*net/wireless/ath9k/xmit.c*).

Additional vendor specific elements can be added to the end beacons and probe response frames of an AP created by *hostapd*, by appending to the configuration file the option *vendor_elements* and setting a valid value on hexadecimal format. *Hostapd* then passes the value to the beacon structure. Like in kernel space, *hostapd* stores beacons in the format "Head +Tail", where all information elements that follow the TIM IE are stored in tail. The function *wpa_driver_nl80211_set_ap* then handles the structure to kernel space. A generalized diagram of beacon information flow from *hostapd* to kernel space is shown in Figure 29.

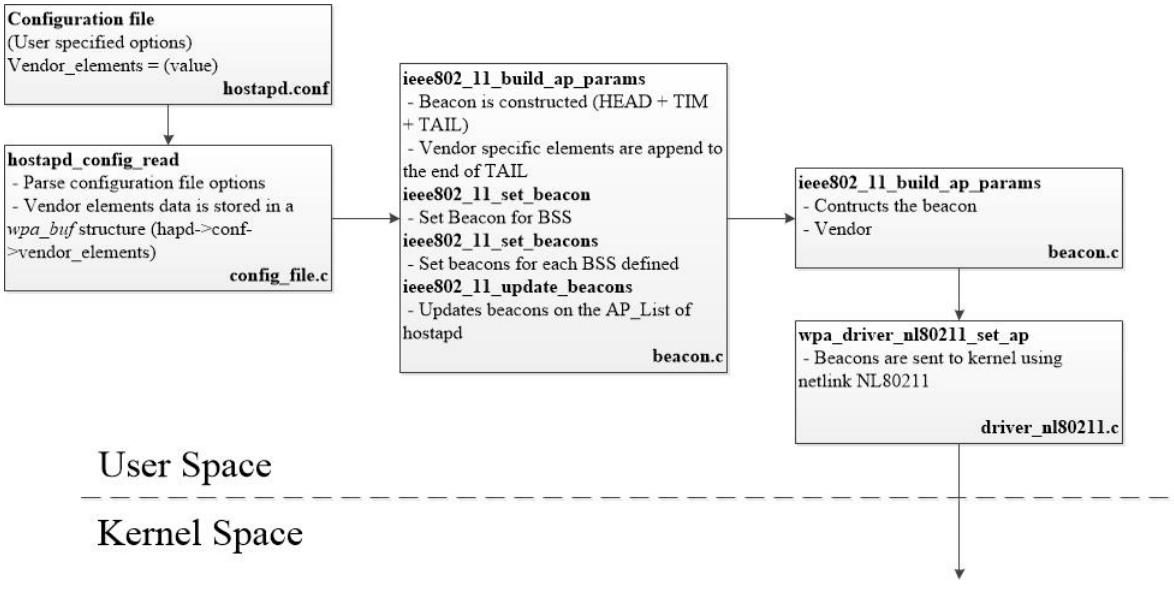


Figure 29 – Beacon set up in hostapd

The beacon template, sent from user space, is handled by nl80211 in kernel space. Contained in a *beacon_parameters* structure, as defined in *cfg80211*, which format is compliant with the one set on hostapd, head and tail, this structure is then associated with the AP structure *ieee80211_if_ap*. The driver, Ath9k, upon each hardware interruption will refer to it, to generate the beacon, and set the proper sequence number to therefore send it. This process is further detailed in Figure 30.

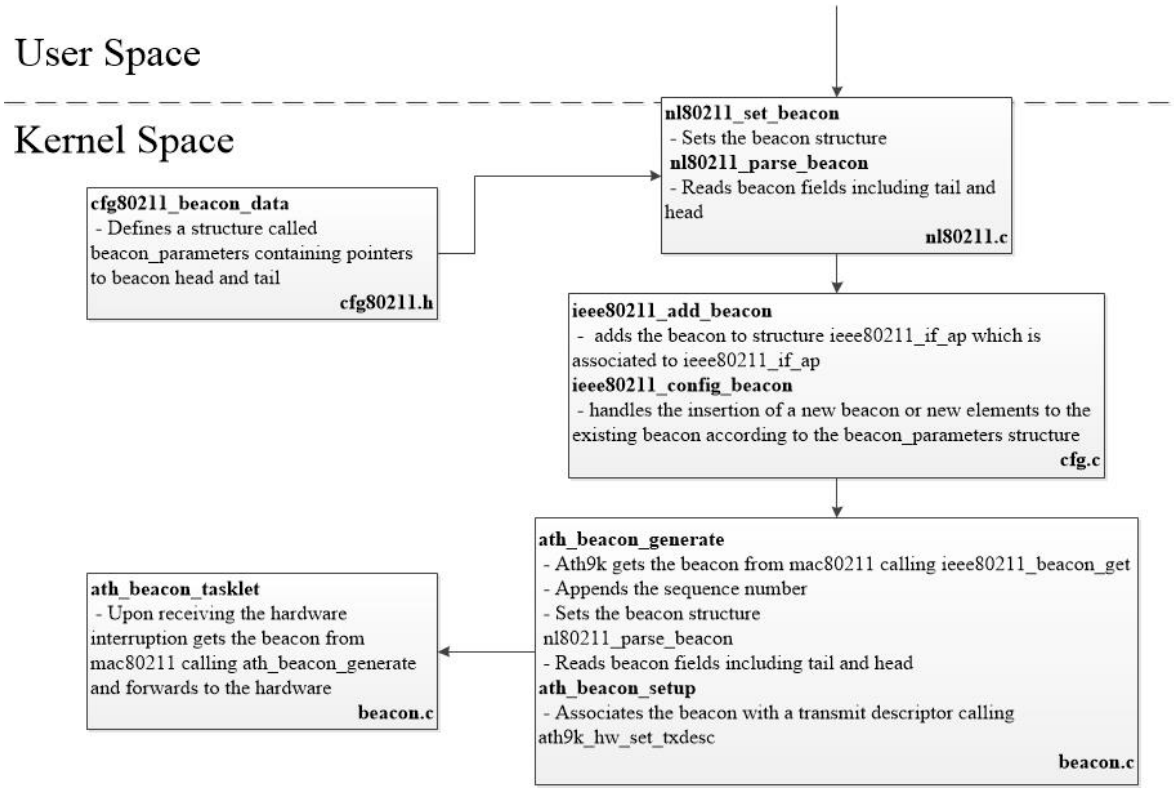


Figure 30 – Beacons in kernel space

The disadvantage of such approach is that the value of the vendor specific element would have to be static during the entire operation of the access point, imposing a strict limitation to control information within the network. Although *hostapd* provides the mechanisms to reload the configuration file, without restarting the daemon by issuing a SIGHUP signal to it, the BSS is restarted which could, in our network, result in full loss of connectivity in the downstream neighbors.

Modifying the beacons is also possible from kernel space, by modifying values in the appropriate structures. Although this would require complex and minute procedures since *vendor_elements* are contained inside multiple structure levels.

A recent release of *hostapd* package enables the addition and modification of vendor elements in beacons directly from the control interface. Receiving the set command from the control interface, *hostapd* repeats the whole process concerning the beacon construction and kernel delivery. This also has other advantages. The network metrics can be appended to real beacons and so minimize control overhead. The access point does not need to be restarted while the modification occurs, so links do not break. This is achievable from user space and do not require complex procedures. We use this mechanism to insert our network metrics into beacons issued by mesh nodes.

```
...
#ifndef CONFIG_CTRL_IFACE_DIR
#define CONFIG_CTRL_IFACE_DIR "/var/run/hostapd"
#endif /* CONFIG_CTRL_IFACE_DIR */
#define OUI "feff00"
#define BEACON_INFO_SIZE 256

static const char *ctrl_iface_dir = CONFIG_CTRL_IFACE_DIR;

static char *ctrl_ifname;
static struct wpa_ctrl *ctrl_conn;
...
int set_beacon_ie221(char *ifname, int hops, int chnl_lst[],int
parent_chnl){

    int ret = 0, i = 0;
    char beacon_info[25];

    sprintf(beacon_info,"dd%02xfeff0001%02x",5+hops,hops);
    // Set tree channel usage info
    for(i=0;i<(hops-1);i++){
        sprintf(beacon_info,"%s%02x",beacon_info,chnl_lst[i]);
    }

    if(!gw)
        sprintf(beacon_info,"%s%02x",beacon_info,parent_chnl);
    ctrl_ifname = ifname;
    ctrl_conn = open_connection(ctrl_ifname);
    if(ctrl_conn){
        printf("\nConnection with hostapd established");
        // Test connection to the control interface. Wait Reply "PONG"
        ret = hostapd_cmd_ping(ctrl_conn);
        // Each SET expects an "OK" on success
        ret = hostapd_cmd_set(ctrl_conn, "vendor_elements", beacon_info);
    }
    else
        printf("\nUnable to open connection with Hostapd");
    close_connection();
    return ret;
}
```

Code Excerpt 2 – Hostapd Set Vendor Element main function

The *set_beacon_ie221* function is in charge of the information element construction and its attachment to the beacon. This function receives as input the name of the interface used to

broadcast beacons, the node number of hops to the gateway, the channel list issued by a parent, if any, and the parent channel. The last two parameters are in fact irrelevant if the node is the gateway. Data is ported in beacons in hexadecimal format, so while appending the information to the beacon decimal data is also converted to its hexadecimal equivalent. Length parameter varies, since the channel list size is incremented at each hop, being the total value equal to the size of fixed parameters (5 bytes) plus the size of the channel list, equal to the number of hops to the gateway.

After the IE is constructed, the connection is established with hostpad daemon using a Unix domain socket, identified by the interface name, through the *open_connection*. Upon return, the pointer to the *wpa_struct* is used to issue all communication with the daemon. A call to *hostapd_cmd_ping* triggers the pairs PING/PONG, to endorse if hostapd is processing the incoming messages, before embedding the IE in the beacon frame via *hostapd_cmd_set*.

4.3.3. ASSOCIATION PROCESS

Stations with no directed path to the gateway will be passively listening to beacons broadcasted from nearby nodes, filtering those with specific SSID and IE OUI. Scans are performed passively without probing the detected APs, since our IEs are only contained in beacons and not in probe response frames, using the *iw* command and parsing the output. Once a valid set of candidate parents is detected the function handles the list to the *choose_neighbor* function that according to topology information will choose the one that guarantees the shortest path to the gateway. We consider the extent of this criteria to a more refined one, since shortest path doesn't necessarily leads to high quality links, by using a linked list and eliminating candidates at each criterion level. However to prove the effectiveness of the proposed solution the specified topology information is adequate. To break any ties among candidate parents (if at least two candidates are at the same level in hierarchy) the first in the list is chosen. Typically the *iw scan* command outputs a list in decreasing order of Received Signal Strength Indicator (RSSI). By choosing the first from the list we increase the probability of selecting a good quality link.

Interface to neighbor bounding proceeds. Having a parent selected, the MAP must divide its interfaces among up and down links, and select the respective configuration to make connection feasible on the uplink and create a BSS on a distinctive band. Node's configuration is edited making a succession of calls to UCI to manage: interface operating

band, mode and in case of the interface selected to provide connection on the downstream, the channel. The procedure used for channel selection is further explained on the following section. A reboot to the wireless system is then performed so that changes to configurations are applied to interfaces, followed by a sleep system call providing a guard time to stabilize the system before further actions.

Nodes can then associate to parents following the sequence of authentication and association request replies to parent's BSS. A call to *iw connect* triggers such process by identifying the SSID and BSSID of the selected service set.

4.4. CHANNEL ASSIGNMENT

To get the most benefit from nodes equipped with two interfaces and the available spectrum in IEEE 802.11 protocol, a simple channel assignment algorithm was developed which avoids channel reutilization in each branch of the tree.

At start up the gateway will run the Automatic Channel Selection (ACS) algorithm to find the least loaded channel. The ACS is a survey based algorithm that gathers information about noise and channel active, busy, and *tx* time to estimate the interference factor and compute the total interference on channels across a specific band. Then the channel with the least overall interference is chosen. The estimation of this parameters is done according to the following procedure. On a specific band the NIC iterates over the available channels. On each one it spends a total of *active time* during which it could either be transmitting for a *tx time* period or waiting for an opportunity to transmit for a *busy time* period. The formula used to estimate this parameter is shown in (3) [47]. While performing the survey the algorithm also registers noise floor and the minimum value is subtracted to each noise floor registered on each channel. The difference is then used in the coefficient of 2 that reflects the way power decreases in the far-field region.

$$Interference\ Factor = \frac{busy\ time - tx\ time}{active\ time - tx\ time} * 2^{chan_{nf} - band_{\min nf}} \quad (3)$$

To account to the fact that channels on 2.4 GHz band overlap, the algorithm also sums the interference of each channel that is overlapped. Finally, the DOWN-NIC tunes to the channel with least degree of interference.

Each MAP keeps two lists: one with all possible channels to be assigned to the DOWN-NIC and another with each channel respective weight in a scale from zero to one. For simplicity of the algorithm only non-overlapping channels on 2.4 GHz band and 5 GHz are considered.

The channel assignment strategy resumes to two different stages: interface to neighbor binding then channel to interface assignment. At start up, scans are performed listening control messages and parsing its content to find a parent that provides the shortest connectivity (in terms of hop count) to the wired infrastructure. According to the band used by the chosen parent, the interface that supports it is chosen to be the UP-NIC and hence the other interface will be used to connect to children nodes. The channel assigned to the UP-NIC is determined by the parent node, since they must match to guaranty that a connection exists between the peers.

Given that the proposed architecture comprises a different band at each hop, a list of candidate channels is created containing all non-overlapping channels, from the opposite band to the one used by the parent, to be assigned to the DOWN-NIC. The weight of each channel is initially set to the maximum value, one. The algorithm then runs through the channel list provided in parent beacons and reduces the weight of candidate channels that have already assigned, according to the distance from the current node at which they have been used. Each time a channel is found on the list his weight is reduced according to equation (4).

$$weight_{k\_new} = weight_{k\_actual} * \frac{d_k}{d_{hops}} \quad (4)$$

$Weight_{k\_actual}$ represents the current weight of a channel used at d_k hops from the current node which is d_{hops} apart from the gateway. When the new weight assignment operation finishes the heaviest channel is assigned to the DOWN-NIC. To break ties the first channel from the candidate list is chosen.

This simple mechanism aims to improve channel diversified paths while reducing the probability of common channel assignment to links in close range of each other.

We assume that no fairness exists in channel assignment but we also consider that the increasing load in nodes with decreasing distance to the gateway is non-trivial and channels with the least interference degree should be assigned to minimize contention.

4.5. ROUTING

Routing is performed on Layer 2.5, based on WiFIX where messages are forwarded through Eo11 tunnels established between parents and children. Each node incorporates a Linux software Bridge where virtual remote nodes connect, permitting each to have an independent routing table so as to provide abstraction to upper layers. Tunnels are created using TAPs, which are software interfaces that behave as layer 2 Ethernet devices, regarding the upper layers point of view, placed at both ends of the tunnel. Linux packet sockets then enable the insertion and reception of data in the tunnel so as the specification of destination and source addresses. Each TAP is created on a node upon a new connection, so when a children selects a parent it immediately creates a TAP and notifies the him, using the ATCM mechanisms, which in turn will trigger the creation of a TAP on the parent to its children..

One of the main enhancements provided to the WiFIX solution is the absence of control traffic by removing the ATCM mechanism. However, the creation of TAPs resorts to ATCM mechanisms, notifying remote endpoints of new nodes at their parenthood. To make the TAP list consistent and updated as new connections are established and terminated without any explicit message a new method had to be developed.

The table of associated stations to an AP is accessible from user space, so one option could be to perform periodic polls and parsing its content for any modifications. However polling has to be well balanced; the increase in polling periodicity augments the waste in processing time if no modification happened to the table between polls, and low periodicity leads to an increase of time to make the tunnel available to route traffic and increases the probability of packet loss. The other option would be to receive kernel events related to associations and disassociations. This away the MAP list is updated almost simultaneously with the associated stations list avoiding all polling disadvantages. The later was preferred and implemented.

Netlink API provides the means to establish connections between user and kernel space. It also enables processes to multicast or listen to multicast groups of the *netlink* family it is connected to, with no need to implement additional features in kernel space. By subscribing to the nl80211 group of *generic netlink* family the algorithm will listen to the wireless system kernel events and filter them to select only the ones concerning MLME, since we want to get event notifications related to associations and disassociations.

The *mlme_event* is the callback function that controls the behavior of the *generic netlink* socket. Code is shown in Code Excerpt .

```
int mlme_event(struct nl_msg *msg, void *arg){

    struct genlmsghdr *gnlh = nlmsg_data(nlmsg_hdr(msg));
    ...

    nla_parse(tb, NL80211_ATTR_MAX, genlmsg_attrdata(gnlh, 0),
genlmsg_attrlen(gnlh, 0), NULL);

    if (tb[NL80211_ATTR_IFINDEX] && tb[NL80211_ATTR_WIPHY]) {
        if_indextoname(nla_get_u32(tb[NL80211_ATTR_IFINDEX]), ifname);
        printf("\n%s (phy #%d): ", ifname,
nla_get_u32(tb[NL80211_ATTR_WIPHY]));
    }

    switch (gnlh->cmd){

    case NL80211_CMD_NEW_STATION:
        mac_addr_n2a(macbuf, nla_data(tb[NL80211_ATTR_MAC]));
        addDownNeighbour(args->taplistp, nla_data(tb[NL80211_ATTR_MAC]),
args->bridge, &iface[ap_id], args->parentp);
        break;

    case NL80211_CMD_DEL_STATION:
        mac_addr_n2a(macbuf, nla_data(tb[NL80211_ATTR_MAC]));
        checkOldDownNeighbour(args-
>taplistp,nla_data(tb[NL80211_ATTR_MAC]),args->bridge, &iface[ap_id]);
        break;

    case NL80211_CMD_DISCONNECT:
        printf("disconnected:");
        if(tb[NL80211_ATTR_DISCONNECTED_BY_AP])
            printf("by AP");
        else
            printf(" (local request)");
        printf("\n");
        break;
    }

    return 0;
}
```

Code Excerpt 3 – Callback for MLME events

This function starts by getting the data from the message and parsing the attributes contained to obtain the interface where the events were generated. The specific event that triggered the message from the kernel is contained in the payload, as a *command* from nl80211. From the list of all possible commands three are of the most interest: the *NL80211_CMD_NEW_STATION* is issued, at a parent, when connections are established, so a new TAP is consequently added to the TAP list with the MAC address of the newly connected children. The insertion to the list is performed by add*DownNeighbor* function which code is shown in APPENDIX E. Oppositely, when a children disconnects the

command *NL80211_CMD_DEL_STATION* is received from the kernel and the corresponding TAP removed from the list by calling *checkOldDownNeighbor* function Appendix E. The *NL80211_CMD_DISCONNECT* is used whenever a disassociation is received from a parent node. Although, because no mechanisms are already implemented in nodes to send such message, no further action is taken, but a simple message for user acknowledgment. On the node that started the association process the TAP to the parent is created immediately.

Once tunnels are created data is able to flow between endpoints. Before being sent over the real interface, at each hop, data is first forwarded to the virtual learning bridge that will map the port of a TAP (or multiple is destination is the broadcast address) as the next hop, according to the intended destination. WiFIX then encapsulates the packet, setting source to be the current node's MAC address and destination to correspond to the TAP (next hop) address. The interface though which data is sent is selected according to the destination of the next hop. Because each interface is associated with a different level of the tree hierarchy, its selection follows a binary criteria, since the next hop is either the parent or any of the children. Exception occurs when, for example, the destination is unknown and ARP requests must be directed in both ways.

Packets are received on the real interface and forwarded to WiFIX, via the packet socket, to proceed to decapsulation. If the recipient is not the destination, the packet will be forwarded to the next hop following the required process to send a message, and substituting the frame header to match the current link endpoints.

5. EXPERIMENTAL ANALYSIS

The experimental analysis conducted, aimed at validating of the proposed solution. In this section, we indorse the use of the beacons as a valid mean to be used as a transport mechanism for network metrics. Then, we describe the followed procedure to carry out experiments and evaluate the proposed solution according to a specific set of metrics.

5.1. BEACONS AS VALID TRANSPORT MECHANISM

Custom vendor IEs in beacons have already been used in other implementations. In [41] the authors introduce the concept of information embedding in vendor specific IEs and teste application scenarios concerning network selection, *Wi-Fi* advertisements and coupons distribution. In [48] authors purpose the use of beacons for indoor positioning systems. Both proposals aim at providing data exchange without the need for association. Although this is also the main objective of beacons in context of this dissertation, since MAPs which are not connected yet to the infrastructure will read and parse the information provided, we also want to ensure that this mechanism is also flexible enough to support other types of metrics. More precisely, the information contained in beacon's vendor specific elements should be manageable without breaking the already established connections. Therefore the first conducted tests aimed to validate this concept.

A basic connectivity test was performed. It comprised a simple BSS created by one ALIX which ran *hostapd* and the developed program that enables the vendor specific element insertion modification and an associated station. Internet Control Message Protocol (ICMP) request/reply packets were exchanged between AP and STA to monitor the connection. To ensure that *vendor elements* were actually modified, a third station was placed within the BSS area capturing beacons sent from the AP. The architecture is depicted in Figure 31.

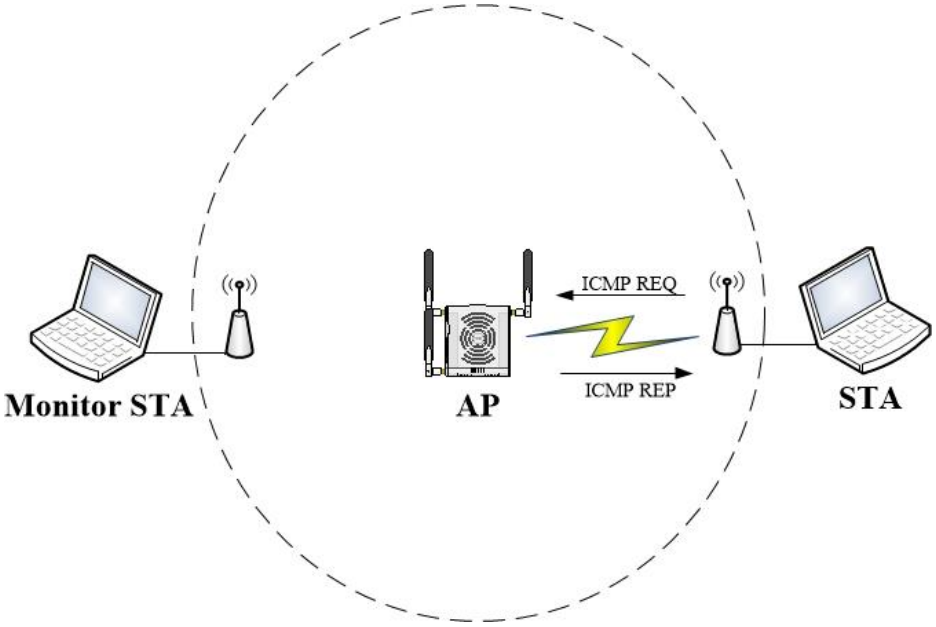


Figure 31 – Beacon Connectivity test architecture

Once the connection was established, between the station and the AP, the flow of ICMP packets is initiated. To test all possible scenarios, no vendor elements were added to beacons at start up. Figure 32 shows a capture of beacons with yet no information inserted.

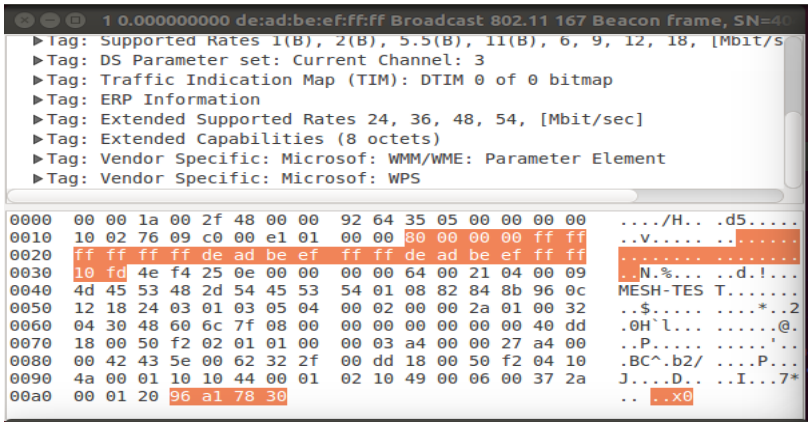


Figure 32 – Beacon with no vendor specific IE

The first vendor IE is then inserted and shown in Figure 33.

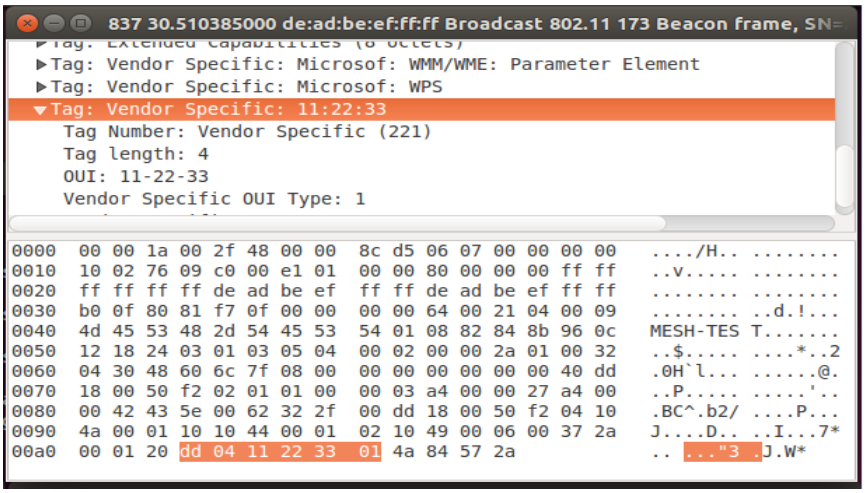


Figure 33 – Beacon with first vendor IEs

The OUI 11-22-33 was defined for sake of this test. The highlighted part in Figure 33 represents the vendor IE constituted by IE id *dd* (hexadecimal of 221), 04 bytes in *length*, OUI and type 01.

The final test aimed to verify the elasticity of the Beacon IE, since metrics may vary in size over time. Thus, the payload size of the IE was increased and decreased in the following beacons, as depicted in Figure 34.

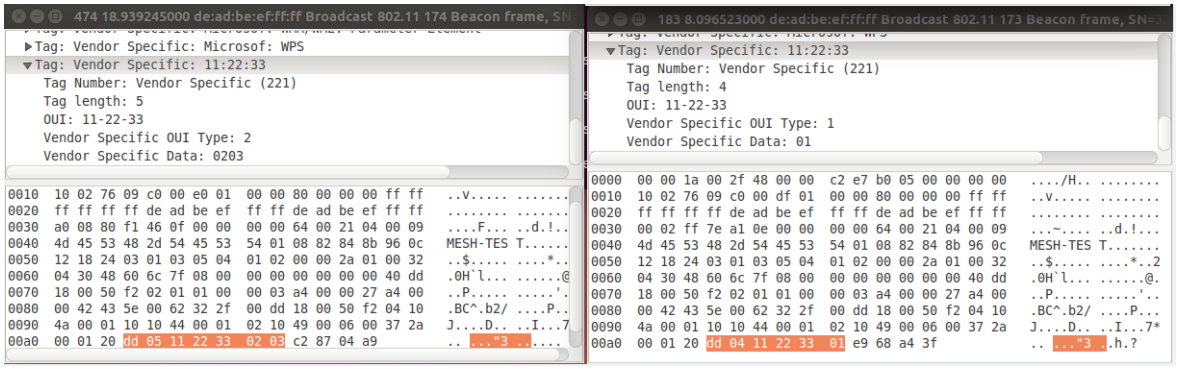


Figure 34 – Beacon vendor IE payload variation

Since the ICMP packet flow was never interrupted during this test, it is possible to validate the vendor specific IE carried in beacons as a flexible transport mechanism for data and applicable in mesh context. The kernel message log was also verified at the STA, to endorse that this procedure is in fact transparent to stations already connected to the BSS. This log is available in Appendix G.

5.2. TESTBED AND PERFORMANCE EVALUATION

In order to evaluate the performance, of the proposed solution conducted a set of experiments and used the original WiFIX solution as a benchmark.

5.2.1. NODE'S HARDWARE

Since the beginning of the development of our testbed, nodes were constituted by single board computers (SBCs), namely *PC Engines Alix System Board 3D3* [49]. An SBC consists of single and small size boards that perform and include most tasks and components usually encountered in ordinary computers, like microprocessor, graphical processor, RAM, GPIO, PCI slots among others. The *Alix* was the logical choice since it fills all requirements for this project and has also been used in many other projects developed at INESC TEC making it very well known among the working group.

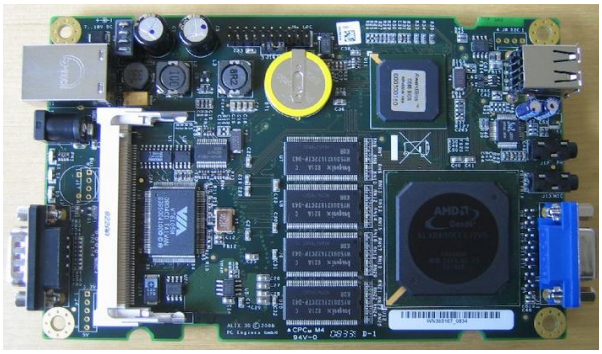


Figure 35 – PC Engines Alix 3d3 [49]

Due to problems (further explained later in *Problems and Limitations*) experienced while performing tests, a TP-LINK Archer C5 v1.2 [50] wireless router was also used. These routers have also been used in INESC TEC projects, namely on the participation on the European project CONFINE (a community network based on wireless mesh) and are equipped with two wireless cards each working on distinctive bands, internal 2.4 GHz and 5 GHz external. It has three built in 2.4GHz antennas and three 5dBi detachable antennas.

Figure 36 – TP-Link Archer c5 v1.2 [50]

Table 1 highlights some key characteristics of both hardware used by our dual-radio dual-band nodes.

Table 1 – Nodes Hardware Specification

	Alix System Board 3D3	**TP-Link Archer C5 AC1200**
Device Type	Single Board Computer	WiFi Router
CPU	500 MHz AMD Geode LX800	720 MHz
Architecture	X86 32 bits	MIPS (ar71xx)
RAM	256 MB DDR RAM	128
Storage	256 MB Flash	16 MB Flash
I/O	DB9 serial port, 2x USB, VGA, audio, headphone out, mic. in	2x USB, 4x 10/100/1000Mbps Lan ports, 1 10/100/1000Mbps WAN Port
Wlan Hardware	2x mini-PCI slots	QCA9558 + QCA9880-BR4A

Two RouterBoard R52n-M mini-PCI network adapters [51] were added to the SBCs to enable the wireless operation, both at 2.4 GHz and 5GHz. These adapters feature an AR9220 chipset.

Figure 37 – MAP's Wireless Network Card [51]

5.2.2. PERFORMANCE METRICS

The experiments conducted in our testbed, aimed to retrieve a set of parameters capable of describing the most important characteristics of the system.

5.2.2.1. Throughput

Throughput represents the amount of data that can be successfully transmitted through a path per unit time. This value is lower than bandwidth, which represents the maximum theoretical rate of data achieved over a communication channel, since the amount of erroneous or non-conforming data is discarded.

To measure throughput, the *Iperf* network test tool was used. It permits the establishment of TCP and UDP flows, between to endpoints, with the objective of measuring throughput, jitter and packet loss. Specifically, the *Iperf* 2.0.5-1 was used, since it is the latest common version to both OpenWrt distributions.

5.2.2.2. Delay/Jitter

Jitter represents a statistical variation in the delay of packet delivery. Packets leave the source of a data stream evenly spaced, but due problems like network congestion or queue congestion, this regular periodicity is affected. As afore stated, *Iperf* also permits to measure such value.

5.2.3. PROBLEMS AND LIMITATIONS

As stated in section 4.1, problems were encountered during experimental analysis, which imposed high constraints to the tests performed and also time consuming repetitions and reboots.

One of such problems is related to DMA crashes, and were already an open issue in the development community for about three years [52] and experienced by some colleagues at INESC TEC. They occurred during UDP tests, more often as the offered bandwidth was increased, causing a node to suddenly loose connection to its parent even on single hop flows, in the current proposed solution, and to suddenly stop listening and transmitting *hellos* when testing the original WiFIX. After a DMA crash, the only way to recover the full node's potential was to reboot the system, since, if only the wireless system was restarted test results were highly affected. Also, and probably related to the DMA problem, some tests had to be repeated very often, since results were highly inconsistent. Nodes also experienced mode deep system crashes, which occurred even if they were just turned on, to which the only workaround was to reboot. These problems, actually, affect both the original WiFIX and the current proposed extension.

From the experience with nodes built on TP-LINK routers, no problems were encountered, so these are preferable for future implementations.

5.2.4. METHODOLOGY AND ACKNOWLEDGMENTS

As the performance of WMN is mainly decided by its backbone network, clients are usually ignored and the corresponding access routers used instead. Our algorithm is applied to the mesh backbone and we aim at providing a simple mechanism to reduce control overhead without disregarding the network overall efficiency. We consider UDP flows from each MAP upwards to the gateway.

The architecture of the testbed used to study the performance of our solution is composed of four nodes and the formed topology is depicted in Figure 38. It consists of an unbalanced binary tree topology, with a maximum of two hops.

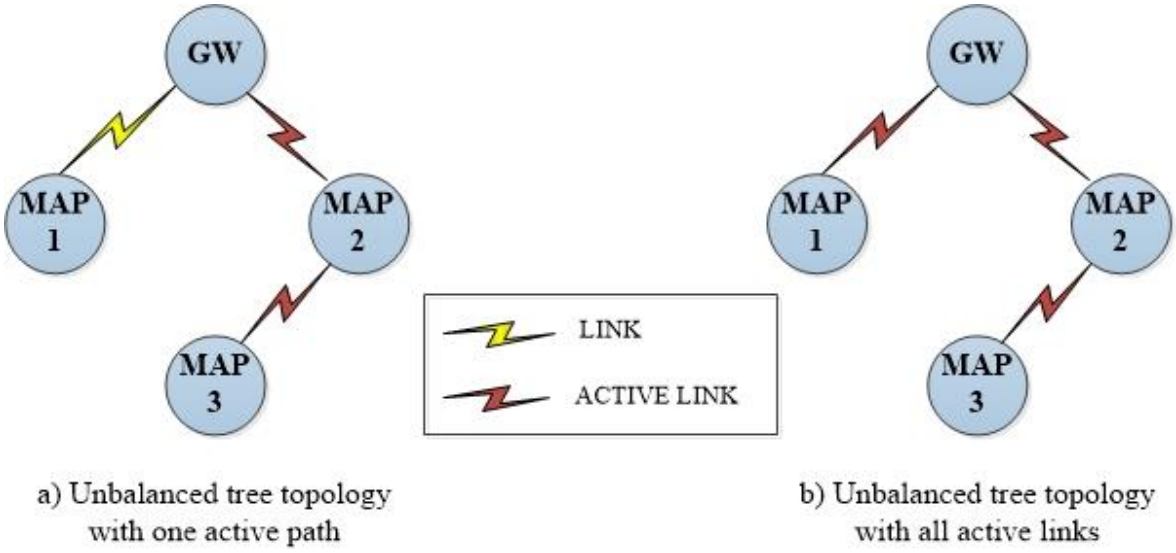


Figure 38 – Testbed setups

We set two test scenarios under the same topology. First, only one flow is active, from the farthest MAP up to the GW (Figure 38a). In the second scenario (Figure 38b), we activate another UDP flow, at one hop distance, with a constant bit rate and keep it on during the whole. Each session ran for 50 seconds, and was repeated five times.

Due to the constraints imposed at this stage and in order to provide coherent and balanced results to all test scenarios, the bit rates were set to the minimum possible of each band. IEEE 802.11 imposes minimum of 1Mb/s on 2.4GHz band (802.11b protocol) and 6Mb/s on 5GHz (802.11a protocol). In spite of introducing unbalanced data rates at each hop and being far from optimal, it was the only possible set to allow to gather some data and establish a comparison between both approaches. Data rates above 1 Mbit/s resulted in an excessive amount crashes and restarting tests. Ideally, the maximum data rates, common to both bands, should be used instead, to establish a fair comparison and provide the upper bound to both solutions.

The original WiFIX runs from a developed bash script on each node, which sets all necessary configurations prior to run it. Since all nodes are in reach of each other, we force the intended topology using the control interface available in the original WiFIX and gateway power variation on the proposed solution, by lowering the transmission power of the GW to the minimum possible while MAP3 is searching for a parent. During tests, the transmission power is set to a fixed value of 20dBm. Beacon interval is set to 100ms in our solution.

5.2.5. NETWORK AUTO CONFIGURATION PROCESS

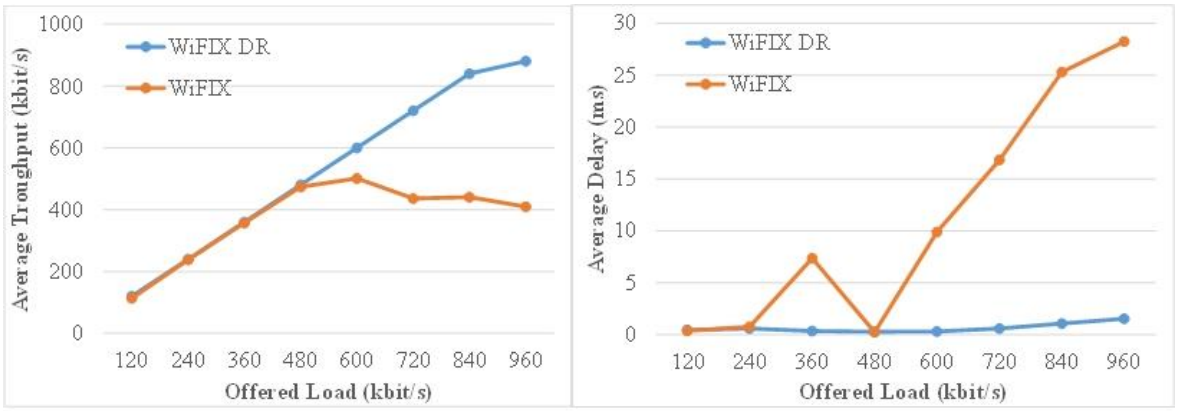
Prior to study the performance of our solution we allowed the network to auto configure following the process described on Chapter 4. A set of screenshots is shown in Appendix H that represent the associations of MAPs according to the topology depicted in Figure 38.

The command *iw station dump* is issued at the GW and lists the stations that are at the first level in the binary tree according to some link parameters. The *iw info* command is issued on each node at the first hop and depicts the information of a specific interface. Here we can see MAP 1 and MAP 2 associated at the GW using channel 6 and that MAP 1 also assigned a channel from a different band (channel 36, 5180 MHz) than the one used to communicate with its parent to its DOWN-NIC.

MAP 3 then joins the network. The list of messages depicted from the association process show that the node starts by performance a scan using wlan0. Upon scanning its surroundings and finding a candidate parent at one hop distance associates to it. The channel assignment to the DOWN-NIC follows. The node choses from the list of non-overlapping channels and decreases the weight of channel 6 since it is already assigned at the GW. The beacon is then constructed using fields *dd07feff0001020624* an issued to *hostapd*. The *iw info* command issued at both interfaces show the channels assigned to both interfaces at MAP 3.

5.2.6. RESULTS AND DISCUSSION

We now present and discuss the obtained results from the performed tests. Starting with the single flow experiment, results are presented in Figure 39. Both solutions present similar throughput and delay concerning offered data rates below 480kbit/s. At this point we reach saturation point in the original WiFIX while WiFIX DR keeps a steady growth with the average throughput matching the offered load till 840 kbit/s. Jitter values, keep this same trend, with WiFIX DR showing much stable values in the whole testing interval.



a) Average throughput versus offered load

b) Average one-way delay versus offered load

Figure 39 – Results with one active path

The major factors influencing throughput degradation in the original WiFIX are related to the half-duplex characteristics of SR deployments. Moreover, CSMA/CA inhibits neighboring parallel transmissions that occur over a common channel. As a consequence, the resultant end-to-end throughput is split over the links that compose the path from MAP 3 to the GW. In fact, it is expected that in multi-hop SRSC networks, throughput decays to *1/n* of the raw channel bandwidth, where *n* is the number of hops [53]. In turn, WiFIX DR introduces much lower overhead on the path to the gateway than its predecessor, resultant either from the removal of explicit control message and by breaking each collision domain at each hop, so the throughput in the two hop link is in fact the same as the as the throughput expected in simple link between AP and STA. To prove the latter statement, we studied the performance of such link following the procedure used to test both WiFIX solutions. Figure 40 depicts the obtained results in a direct comparison on throughput with WiFIX DR.

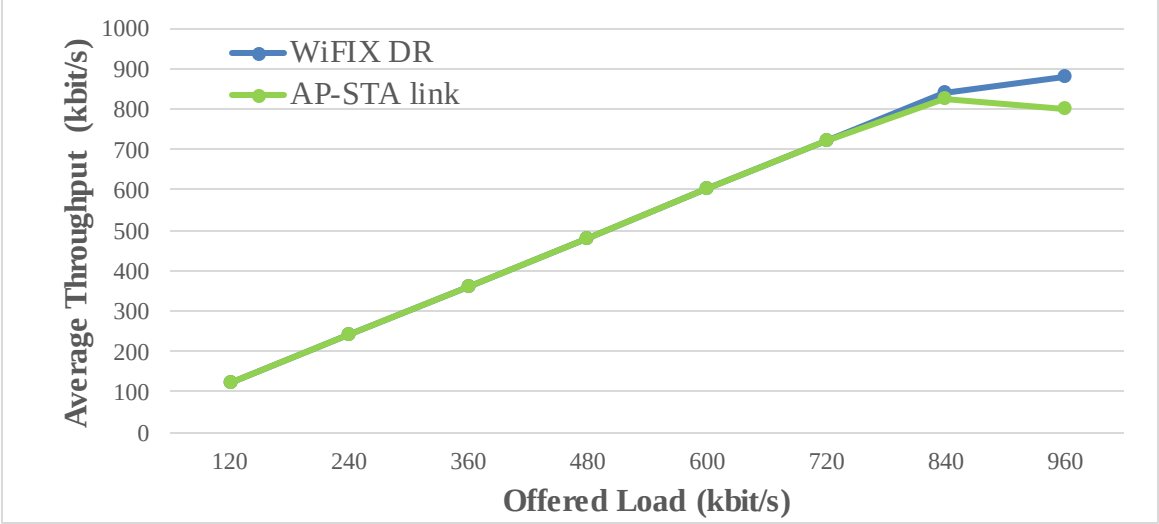
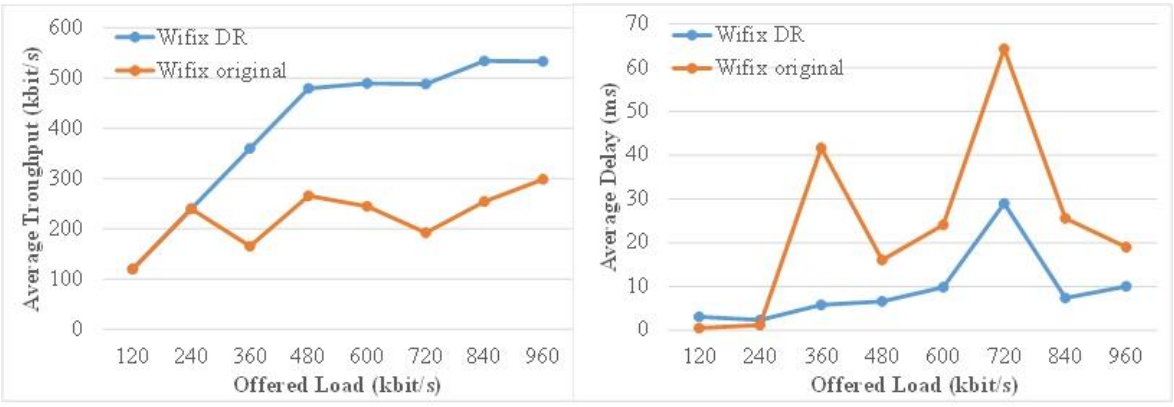


Figure 40 – WiFIX DR at two hops and AP-STA link average throughput

Results on the second scenario are presented in Figure 41. The decay in channel performance comparatively with the first scenario is expected, since now a bottleneck exists with two MAPs competing for an opportunity to transmit at first level in the binary tree. Although, in the original WiFIX data flows at each level will reciprocally interfere and introduce higher overhead at both levels.



a) Average throughput versus offered load

b) Average one-way delay versus offered load

Figure 41 – Results with all active links

Again, comparing WiFIX DR with an AP connected to two STAs, as shown in (Figure 42) we prove that throughput at a two hop distance is only limited by competing flows at the GW.

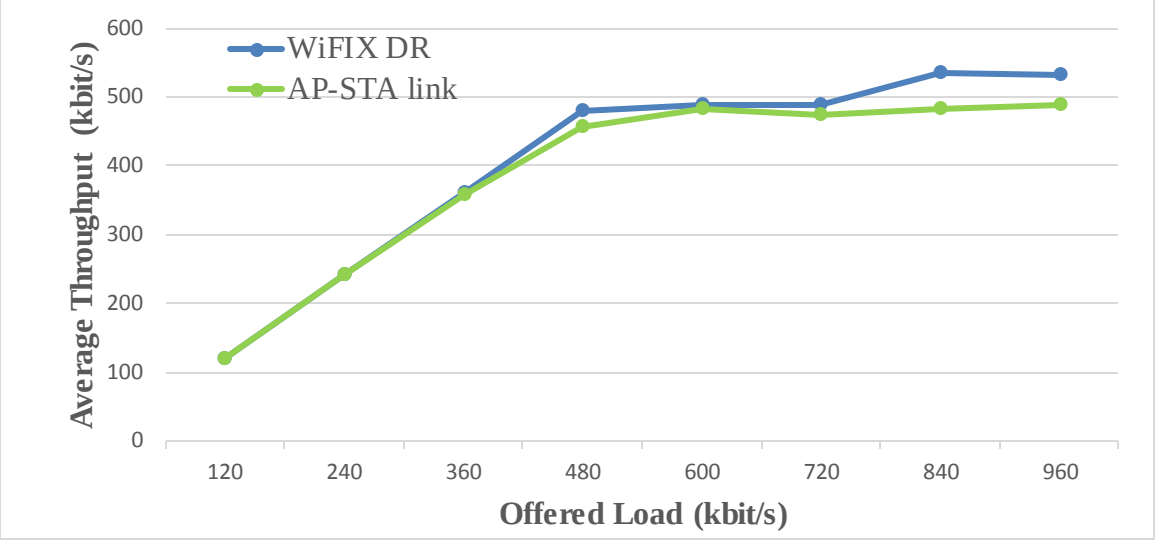


Figure 42 – WiFIX DR all active links and AP - two STAs average throughput

Note that, in spite of some inconsistency of results, we are able to trace the general profile of both performance metrics in the stated scenarios. In fact, the proposed solution outperforms the original WiFIX in both throughput and delay even in a small testbed. The obtained results clearly show the improvement of the extension proposed to WiFIX concerning the scalability of the multi-hop network. Specifically MAPs joining WiFIX DR are broadly aware of interference on upstream links and use this knowledge to minimize the impact of joining the network. Control messages are suppressed and embedded in mechanisms already implemented in IEEE 802.11 based networks, freeing the medium from unnecessary overhead that most state of art solutions impose. Also we achieved this in an inexpensive solution, equipping nodes with only two interfaces and utilizing the full frequency spectrum, available in IEEE 802.11a and IEEE 802.11b/g, for communication purposes. Furthermore, since using WiFIX routing scheme and the inherit 802.1D learning bridges mechanisms, paths are established with data or signaling messages and based on Layer 2.5.

One major weaknesses of the proposed solution, in comparison with other state of art MRMC WMN solutions, is the unawareness of mesh topology and link degradation. Once the topology is established a node will no longer disassociate from the current parent to one that guarantees shortest and better path quality. Although the oscillations result from frequent topology variation, and the collateral impact on the overall performance and channel assignment in the mesh are definitely theme to another thesis. The current channel assignment algorithm might perform well concerning vertical (in the hierarchical sense)

interference, although it fails to reduce the horizontal interference. Hence, nodes at the same hierarchical level will be assigned a common channel and split the available raw bandwidth resulting in a decrease of throughput.

6. CONCLUSIONS AND FUTURE WORK

The purpose of this thesis was to provide a comprehensive study on mesh networks, concerning their weaknesses, challenges and state of art MRMC schemes developed to addressed them. Upon this study it was expected to outline an innovative scheme that enables the extension of a wired infrastructure with improved scalability and performance.

Most state of art solutions use additional packets to exchange control information among MAPs, in order to permit the network to scale or adapt to new conditions. Also some solutions allot one interface or/and channel for control purposes wasting radio resources and frequency spectrum. Some schemes performed routing on Layer 3 while Layer 2.5 provides more flexible solution. Upon this key concepts we developed a new state of art MRMC multi-hop solution that uses the minimal number of interfaces, takes advantages of the mechanisms from IEEE 802.11 networks to broadcast information, to distribute metrics and performs routing on layer 2.5. Additionally, the purposed solution also takes advantage of the provided IEEE 802.11 frequency spectrum on 2.4 GHz and 5 GHz bands to reduce intra-path and cross-board interference.

The challenge of embedding information in beacons was in fact an ambitious one, and was to be best of our knowledge the first time such mechanism has been implemented in a mesh context. WiFIX routing protocol was also successfully adapted to the new context, establishing Eo11 tunnels synchronously with associations by listening to kernel notifications.

The chosen platform to develop our testbed imposed high constraints to performance tests. Still, we were able to tests the performance of the developed solution and prove the effectiveness and improved scalability against the original WiFIX.

6.1. CONTRIBUTIONS

Concluded the course of our study we state the accomplished contributions. We provided a survey on the current state of art solutions that address the challenges of MRMC WMNs concerning those that jointly address Channel Assignment and Routing or that address each independently. Identified a gap in the current state of art, we proposed a new cost-effective distributed multi-hop MRMC scheme in a JCAR solution routing over Layer 2.5. We presented a new transport mechanism for mesh network metrics that reduces the overhead introduced from control messages and validated its use and flexibility. Moreover, we extended WiFIX to support this new metrics transport mechanism and dual-radio nodes. Finally, we concluded this work with a proof-of-concept and evaluation comparing it with the original WiFIX

6.2. FUTURE WORK

For future work, we consider of fundamental importance:

- A new testbed has to be assembled, with MAPs running on different hardware and Operating System. The constraints imposed by ALIXs and OpenWRT 14.04 became the major bottleneck of our solution. One option could be to use the TP-LINK routers since they present no problems while used. Also on the stable release of OpenWRT Caos Calmer rc3 the TX & DMA bugs are announced to be fixed [54].

- Having a stable testbed, new performance tests should be conducted on a denser testbed and against other MRMC WMN solutions.

Although throughout the development of this work we have mostly invested our time in some other traits of the proposed solution, we consider that the following points should not be disregarded from future research and implementations:

- Enable the self-reorganization of the network. Although static deployments don't suffer from oscillations of purely dynamic networks, a balanced solution could benefit from the advantages of both. Nevertheless, the impact and metrics to use in this scheme should be well weighted.

- The improvement of channel assignment strategy in order to guarantee a better interference balance in the entire network.

References

[1] J. Mamede. COMOV L. Class Lecture, Topic "Wireless LAN", Instituto Superior de Engenharia do Porto, Jan. 01, 2015.

[2] K. Xu, M. Gerla, and S. Bae, "How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks," presented at the Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE, 2002, vol. 1, pp. 72–76 vol.1.

[3] A. Raniwala, K. Gopalan, and T. Chiueh, "Centralized Channel Assignment and Routing Algorithms for Multi-channel Wireless Mesh Networks," *SIGMOBILE Mob Comput Commun Rev*, vol. 8, no. 2, pp. 50–65, Apr. 2004.

[4] "EN 301 893 - V1.7.2 - Broadband Radio Access Networks (BRAN); 5 GHz high performance RLAN; Harmonized EN covering the essential requirements of article 3.2 of the R&TTE Directive." .

[5] H. Skalli, S. Ghosh, S. K. Das, L. Lenzini, and M. Conti, "Channel Assignment Strategies for Multiradio Wireless Mesh Networks: Issues and Solutions," *Commun. Mag. IEEE*, vol. 45, no. 11, pp. 86–95, Nov. 2007.

[6] R. Riggio, T. Rasheed, S. Testi, F. Granelli, and I. Chlamtac, "Interference and traffic aware channel assignment in WiFi-based wireless mesh networks," *Ad Hoc Netw.*, vol. 9, no. 5, pp. 864–875, Jul. 2011.

[7] Y. Ding, K. Pongaliur, and L. Xiao, "Channel Allocation and Routing in Hybrid Multichannel Multiradio Wireless Mesh Networks," *Mob. Comput. IEEE Trans. On*, vol. 12, no. 2, pp. 206–218, Feb. 2013.

[8] M. Doraghinejad, H. Nezamabadi-pour, and A. Mahani, "Channel assignment in multi-radio wireless mesh networks using an improved gravitational search algorithm," *J. Netw. Comput. Appl.*, vol. 38, pp. 163–171, Feb. 2014.

[9] S. A. Makram and M. Güneş, "Distributed channel assignment for multi-radio wireless mesh networks," presented at the Computers and Communications, 2008. ISCC 2008. IEEE Symposium on, 2008, pp. 272–277.

[10] A. Naveed, S. S. Kanhere, and S. K. Jha, "Topology Control and Channel Assignment in Multi-Radio Multi-Channel Wireless Mesh Networks," presented at the Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE International Conference on, 2007, pp. 1–9.

[11] K. Athota and A. Negi, "A topology preserving cluster-based channel assignment for wireless mesh networks," *Int. J. Commun. Syst.*, vol. 28, no. 12, pp. 1862–1883, Aug. 2015.

[12] H. Cheng, N. Xiong, A. V. Vasilakos, L. Tianruo Yang, G. Chen, and X. Zhuang, "Nodes organization for channel assignment with topology preservation in multi-radio wireless mesh networks," *Ad Hoc Netw.*, vol. 10, no. 5, pp. 760–773, Jul. 2012.

[13] J. Wang, W. Shi, K. Cui, F. Jin, and Y. Li, "Partially overlapped channel assignment for multi-channel multi-radio wireless mesh networks," *EURASIP J. Wirel. Commun. Netw.*, vol. 2015, no. 1, pp. 1–12, Feb. 2015.

[14] M. K. Marina, S. R. Das, and A. P. Subramanian, "A topology control approach for utilizing multiple channels in multi-radio wireless mesh networks," *Comput. Netw.*, vol. 54, no. 2, pp. 241–256, Feb. 2010.

[15] J.-W. Lin and S.-M. Lin, "A weight-aware channel assignment algorithm for mobile multicast in wireless mesh networks," *J. Syst. Softw.*, vol. 94, pp. 98–107, Aug. 2014.

[16] R. Campos, R. Duarte, F. Sousa, M. Ricardo, and J. Ruela, "Network infrastructure extension using 802.1D-based wireless mesh networks," *Wirel. Commun. Mob. Comput.*, vol. 11, no. 1, pp. 67–89, Jan. 2011.

[17] D. S. Nandiraju, N. S. Nandiraju, and D. P. Agrawal, "Adaptive state-based multi-radio multi-channel multi-path routing in Wireless Mesh Networks," *Pervasive Mob. Comput.*, vol. 5, no. 1, pp. 93–109, Feb. 2009.

[18] S.-M. He, D.-F. Zhang, K. Xie, H. Qiao, and J. Zhang, "Channel Aware Opportunistic Routing in Multi-Radio Multi-Channel Wireless Mesh Networks," *J. Comput. Sci. Technol.*, vol. 29, no. 3, pp. 487–501, May 2014.

[19] Y. Peng, Y. Yu, L. Guo, D. Jiang, and Q. Gai, "An efficient joint channel assignment and QoS routing protocol for IEEE 802.11 multi-radio multi-channel wireless mesh networks," *J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 843–857, Mar. 2013.

[20] Y. Zhao, H. Wang, Y. Li, and H. Song, "A Multi-channel Routing Protocol for Dual Radio Wireless Networks," presented at the High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on, 2013, pp. 2271–2276.

[21] D. Seither, A. König, and M. Hollick, "Routing performance of Wireless Mesh Networks: A practical evaluation of BATMAN advanced," presented at the Local Computer Networks (LCN), 2011 IEEE 36th Conference on, 2011, pp. 897–904.

[22] A. Quartulli and R. Lo Cigno, "Client announcement and Fast roaming in a Layer-2 mesh network," University of Trento, Departmental Technical Report, Oct. 2011.

[23] A. Raniwala, K. Gopalan, and T. Chiueh, "Centralized Channel Assignment and Routing Algorithms for Multi-channel Wireless Mesh Networks," *SIGMOBILE Mob Comput Commun Rev*, vol. 8, no. 2, pp. 50–65, Apr. 2004.

[24] A. Dhananjay, H. Zhang, J. Li, and L. Subramanian, "Practical, Distributed Channel Assignment and Routing in Dual-radio Mesh Networks," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, New York, NY, USA, 2009, pp. 99–110.

[25] A. Raniwala, Tzi-cker Chiueh, "Architecture and algorithms for an IEEE 802.1 1 -based multi-channel wireless mesh network," 2005, vol. 3, pp. 2223–2234.

[26] R. Koshy and L. Ruan, "A Joint Radio and Channel Assignment (JRCA) Scheme for 802.11-Based Wireless Mesh Networks," presented at the GLOBECOM Workshops, 2009 IEEE, 2009, pp. 1–6.

[27] A. Pal and A. Nasipuri, "JRCA: A joint routing and channel assignment scheme for wireless mesh networks," presented at the Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International, 2011, pp. 1–8.

[28] P. Kyasanur and N. H. Vaidya, "Routing and link-layer protocols for multi-channel multi-interface ad hoc wireless networks," *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 10, no. 1, pp. 31–43, Jan. 2006.

[29] S. Pollak, V. Wieser, and A. Tkac, "A channel assignment algorithm for wireless mesh networks with interference minimization," presented at the Wireless and Mobile Networking Conference (WMNC), 2012 5th Joint IFIP, 2012, pp. 17–21.

[30] K. Ramachandran, I. Sheriff, E. M. Belding, and K. C. Almeroth, "A multi-radio 802.11 mesh network architecture," *Mob. Netw. Appl.*, vol. 13, no. 1–2, pp. 132–146, Jan. 2008.

[31] V. Gardellin, S. K. Das, L. Lenzini, C. Cicconetti, and E. Mingozzi, "G-PaMeLA: A divide-and-conquer approach for joint channel assignment and routing in multi-radio multi-channel wireless mesh networks," *J. Parallel Distrib. Comput.*, vol. 71, no. 3, pp. 381–396, Mar. 2011.

[32] D. Wu, S.-H. Yang, L. Bao, and C. H. Liu, "Joint multi-radio multi-channel assignment, scheduling, and routing in wireless mesh networks," *Wirel. Netw.*, vol. 20, no. 1, pp. 11–24, Apr. 2013.

[33] J. Wellons and Y. Xue, "The robust joint solution for channel assignment and routing for wireless mesh networks with time partitioning," *Ad Hoc Netw.*, vol. 13, Part A, pp. 210–221, Feb. 2014.

[34] J. J. Gálvez and P. M. Ruiz, "Joint link rate allocation, routing and channel assignment in multi-rate multi-channel wireless networks," *Ad Hoc Netw.*, vol. 29, pp. 78–98, Jun. 2015.

[35] M. Gast. *802.11 Wireless Networks: The Definitive Guide*. (2nd Edition) O'Reilly, Apr. 2005. pp. 46 -54

[36] "rtnetlink(7) - Linux manual page." [Online]. Available: http://man7.org/linux/man-pages/man7/rtnetlink.7.html. [Accessed: 21-Oct-2015].

[37] "netlink(7) - Linux manual page." [Online]. Available: http://man7.org/linux/man-pages/man7/netlink.7.html. [Accessed: 21-Oct-2015].

[38] J. Berg, "About mac80211 [Linux Wireless]." [Online]. Available: https://wireless.wiki.kernel.org/en/developers/Documentation/mac80211. [Accessed: 21-Oct-2015].

[39] “Index of /barrier_breaker/14.07/x86/generic/.” [Online]. Available: https://downloads.openwrt.org/barrier_breaker/14.07/x86/generic/. [Accessed: 15-Oct-2015].

[40] “Index of /chaos_calmer/15.05/ar71xx/generic/.” [Online]. Available: https://downloads.openwrt.org/chaos_calmer/15.05/ar71xx/generic/. [Accessed: 15-Oct-2015].

[41] R. Chandra, J. Padhye, L. Ravindranath, and A. Wolman, “Beacon-Stuffing: Wi-Fi without Associations,” presented at the Mobile Computing Systems and Applications, 2007. HotMobile 2007. Eighth IEEE Workshop on, 2007, pp. 53–57.

[42] V. Gupta and M. K. Rohil, “Bit-Stuffing in 802.11 Beacon Frame: Embedding Non-Standard Custom Information.”

[43] “IEEE Standards OUI List.” [Online]. Available: http://standards-oui.ieee.org/oui.txt. [Accessed: 02-Oct-2015].

[44] “11-02-0592-00-000e-wireless-multimedia-enhancements-wme-phase-1.doc.” .

[45] “PyLorcon2.c - Google Project Hosting.” [Online]. Available: https://code.google.com/p/pylorcon2/source/browse/trunk/PyLorcon2.c?r=8. [Accessed: 15-Oct-2015].

[46] “pylorcon2 · GitHub.” [Online]. Available: https://github.com/tom5760/pylorcon2. [Accessed: 15-Oct-2015].

[47] “ACS (Automatic Channel Selection) - Linux Wireless.” [Online]. Available: https://wireless.wiki.kernel.org/en/users/documentation/acs. [Accessed: 02-Oct-2015].

[48] O. Abu Oun, C. Bloch, and F. Spies, “Indoor positioning using CoLDE: An IEEE 802.11 connectionless extension,” presented at the Indoor Positioning and Indoor Navigation (IPIN), 2014 International Conference on, 2014, pp. 491–500.

[49] “PC Engines alix3d3 product file.” [Online]. Available: http://pcengines.ch/alix3d3.htm. [Accessed: 27-Oct-2015].

[50] TP-LINK. *Archer C5 1.0 - Archer_C5_V1_Datasheet*. [Online]. Available: http://www.tp-link.com/resources/document/Archer_C5_V1_Datasheet.pdf [Accessed: 15-Oct-2015]

[51] “RouterBoard.com : R52nM.” [Online]. Available: http://routerboard.com/R52NM. [Accessed: 27-Oct-2015].

[52] “#11862 (ath: Failed to stop TX DMA, queues=0x004) – OpenWrt.” [Online]. Available: https://dev.openwrt.org/ticket/11862. [Accessed: 14-Sep-2015].

[53] Y. Zhang, J. Luo, H. Hu. *Wireless Mesh Networking*. Boca Raton, NY. Auerbach Publications, 2006, pp. 8 - 25.

[54] “OpenWrt.” [Online]. Available: https://openwrt.org/. [Accessed: 01-Nov-2015].

Appendix A

Appendix A.1

The following table highlights the some of the most important characteristics of CA solutions presented in State of Art.

Table 1 – Summary of CA solutions

Solução	Coordination	NICs per Node	Control dedicated NIC/Channel	COTS	Objective	Method	Topology Type	Results
Mestic	Centralized	>1	YES	YES	Improve throughput and maintain topological connectivity	Rank nodes and assign least interference channels	Fixed Tree-based	Simulation ns2
ITACA	Centralized	NS	YES	YES	Minimum Interference	Assign Channels accounting for interface with a ranking function and traffic considering flow fluctuations	Hierarchical Tree-based	Experimental + Simulation ns2
ADCA	Distributed	>1	YES	YES	Adapt to traffic variations with low OH	Split time into control and data intervals. Fixed and dynamic NICs	Tree-based	Simulation ns-2
IGSA	NS	NS	NS	NS	Minimize interference and maximize throughput	Uses a GSA to find the best solution. DLS keeps the best solution	Fixed Tree-based	MATLAB
CCA	Centralized	NS	YES	NS	Distributed CA to reduce interference and improve throughput	CHH performs CA	Cluster	Simulation ns-2
COMTAC	Distributed	>1	YES	YES	Improve throughput reducing interference	Model interference in default and non-default channels	Cluster	Simulation Qualnet

CBCA	Centralized Clustering / Distributed CA	>1	YES	NS	Improve throughput reducing interference	Create a hierarchical cluster set	Cluster	Simulation ns-2
LaSo	Distributed	NS	Yes	NS	Improve throughput reducing interference	Access control using latin squares and graph coloring	Hierarchical Cluster	Simulation QualNet 4.5
DPSO-CA	Centralized	>1	YES	NS	Minimum Interference	DPSO and genetic algorithms	NS	NS
POCA	Centralized / Distributed	NS	NS	YES	Link optimization	Estimate the impact of POCs by the reduced IR factor	Fixed Tree	Simulation NS-3.19
CAMF	Distributed	NS	YES	YES	Minimum interference on transmission	Use forwarding weight parameter to estimate interference before Tx	Hierarchical tree	Simulation OPNET
CLICA	Centralized	>1	NO	YES	Minimize interference or Improve traffic flows	Use fewer NICs on each node and assign channels to nodes in a given priority	Fixed-Tree	Simulation NS-2

Appendix A.2

The following table highlights the some of the most important characteristics of routing solutions presented in State of Art.

Table 2 – Summary of routing solutions

Solution	Coordination	NICs per Node	Control dedicated Channel/NIC	COTS	Objective	Method	Layer	Topology Type	Metrics	Results
ASMRP	Distributed	3	NO	YES	Distribute load among less congested links	Maintaining a set of possible routes and measuring queues size over time	NS	Hierarchical	Average load and link capacity	Simulation (ns-2)
CAOR	Distributed	>1	NO	YES	Find an OR path to explore MR and improve performance	Weight cost possible forwarders to a destination over multiple channels	NS	Fixed tree	MEATT – considers the shortest path	Lingo
CLQ-OLSR	Distributed	>1	NS	YES	Provide high QoS path for multimedia purposes	Exchange bandwidth and topology information	NS	NS	Shortest hop + Bandwidth	Simulation (Qualnet)
WiFIX	Distributed	1	NO	YES	Extend the wired infrastructure with a single message	Create Eo11 tunnels and with learning bridges at tunnel endpoints	2.5	Hierarchical	Hop count	Experimental
MCDR	Distributed	2	NO	NO (DIM MAC)	Find channel diversified routes with minimum delay	Distribute traffic through channels weighting channel switching times	2.5	NS	Channel Switching times + ETT + Load degree + WCETT-LB	Simulation (GlomoSim)
BATMAN-adv	Distributed	NS	NO	YES	Locally optimize routes	Network Flooding OGM messages	2.5	NS	Tramission Quality	Experimental

Appendix A.3

The following table highlights the some of the most important characteristics of JCAR solutions presented in State of Art.

Table 3 – Summary of JCAR solutions

Solution	Coordination	Priority	NICs per Node	Control dedicated NIC/Channel	COTS	Iterative Solution	Topology Type	Routing	Layer	Results
ROMA	Distributed	CA	2/3	NO	YES	NO	Hierarchical	ETT + External Load	3	Experimental
Hyacinth	Distributed	CA	NS	NO	YES	NO	NS	Shortest Hop + gw link cap. + Path cap.	3	Simulation (ns-2) + Experimental
LACA	Centralized	Routing	NS	NO	YES	YES	NS	Shortest Hop or Random Multi-path	NS	Simulation (ns-2) + Practical
JRCA	Centralized	Routing	>1	YES	NS	YES	NS	Probability of Success + delay	NS	Simulation (ns-2)
LL	Distributed	CA	NS	NO	YES	NO	NS	MCR	3	Qualnet v3.6
FRCA	Centralized	Routing	2-8	NO	NS	YES	NS	Shortest Path	NS	Simulation (ns-2)
MCUBE	Centralized	CA	>1	YES (channel)	YES	NO	NS	WCETT	3	Experimental
G-PaMeLa	Centralized	CA and Routing	NS	YES (NIC)	YES	NO	NS	Physical Top, Nº of NICs Signal Pw, Channel rate, Traffic load	NS	Simulation (ns-2)
M4	Distributed	Routing	NS	YES	NS	NO	Cluster	Shortest path + RSSI/SNR + MAC OH + Queue Size	NS	Simulation (QualNet 4.5)
RCART	Centralized	Routing	NS	NS	NS	YES	NS	Traffic infinitesimally divided through multiple GWs and considering history	NS	NS
JMR	Distributed	Link rate allocation	NS	NO	NS	NO	Binary tree with ringnodes	Distributed source routing	NS	Simulation (ns-3)

Appendix B.

Makefile to compile for both x86 and MIPS platforms.

```
#
#   MAKEFILE WIFIX_DR
#
BASE_DIR = /home/pedro/GoogleDrive/TeseCrossComp
CFLAGS = -Wall -g

#########################################################################
###################################
#
#           x86 Build
#
#########################################################################
###################################
INCLUDE_HOSTAPD = $(BASE_DIR)/hostap_x86/src
LIBRARY_HOSTAPD = $(BASE_DIR)/hostap_x86/src/common
OBJS_HOSTAPD=$(INCLUDE_HOSTAPD)/common/wpa_ctrl.o
$(INCLUDE_HOSTAPD)/utils/os_unix.o
HEADERS_HOSTAPD=$(INCLUDE_HOSTAPD)/common/wpa_ctrl.h
$(INCLUDE_HOSTAPD)/utils/os.h

OPENWRT_DIR = /home/pedro/OpenWrt-SDK-x86-for-linux-x86_64-gcc-4.8-
linaro_uClibc-0.9.33.2

CC = $(OPENWRT_DIR)/staging_dir/toolchain-i386_i486_gcc-4.8-
linaro_uClibc-0.9.33.2/bin/i486-openwrt-linux-uclibc-gcc

# VARIABLES
INCLUDE_NORMAL = $(OPENWRT_DIR)/staging_dir/toolchain-i386_i486_gcc-
4.8-linaro_uClibc-0.9.33.2/include
INCLUDE_OPENSSL = $(OPENWRT_DIR)/build_dir/target-i386_i486_uClibc-
0.9.33.2/openssl-1.0.2d/include
INCLUDE_LIBNL = $(OPENWRT_DIR)/staging_dir/target-i386_i486_uClibc-
0.9.33.2/usr/include/libnl3/

LIBRARY_NORMAL =  $(OPENWRT_DIR)/staging_dir/toolchain-i386_i486_gcc-
4.8-linaro_uClibc-0.9.33.2/lib
LIBRARY_OPENSSL = $(OPENWRT_DIR)/build_dir/target-i386_i486_uClibc-
0.9.33.2/openssl-1.0.2d
LIB_DIR = $(OPENWRT_DIR)/staging_dir/target-i386_i486_uClibc-
0.9.33.2/usr/lib

#########################################################################
#
#           MIPS ar71xx Build
#   Uncomment the following part to generate a MIPS compatible file
#########################################################################
#INCLUDE_HOSTAPD = $(BASE_DIR)/hostap_mips/src
#LIBRARY_HOSTAPD = $(BASE_DIR)/hostap_mips/src/common
#OBJS_HOSTAPD=$(INCLUDE_HOSTAPD)/common/wpa_ctrl.o
$(INCLUDE_HOSTAPD)/utils/os_unix.o
#HEADERS_HOSTAPD=$(INCLUDE_HOSTAPD)/common/wpa_ctrl.h
$(INCLUDE_HOSTAPD)/utils/os.h
```

```
#
#OPENWRT_DIR = /home/pedro/OpenWrt-SDK-15.05-ar71xx-generic_gcc-4.8-
linaro_uClibc-0.9.33.2.Linux-x86_64
#CC = $(OPENWRT_DIR)/staging_dir/toolchain-mips_34kc_gcc-4.8-
linaro_uClibc-0.9.33.2/bin/mips-openwrt-linux-uclibc-gcc
#
##VARIABLES
#INCLUDE_NORMAL = $(OPENWRT_DIR)/staging_dir/toolchain-mips_34kc_gcc-
4.8-linaro_uClibc-0.9.33.2/include
#INCLUDE_OPENSSL = $(OPENWRT_DIR)/build_dir/target-mips_34kc_uClibc-
0.9.33.2/openssl-1.0.2d/include
#INCLUDE_LIBNL = $(OPENWRT_DIR)/staging_dir/target-mips_34kc_uClibc-
0.9.33.2/usr/include/libnl3/
#
#LIBRARY_NORMAL =  $(OPENWRT_DIR)/staging_dir/toolchain-mips_34kc_gcc-
4.8-linaro_uClibc-0.9.33.2/lib
#LIBRARY_OPENSSL = $(OPENWRT_DIR)/build_dir/target-mips_34kc_uClibc-
0.9.33.2/openssl-1.0.2d
#LIB_DIR = $(OPENWRT_DIR)/staging_dir/target-mips_34kc_uClibc-
0.9.33.2/usr/lib

################################### DO NOT
EDIT############################################################
OBJS = ./src/*.o
DEPS = include/*.h
SOURCE = src/bcast.c src/bridgeutils.c src/forward.c src/mainloop.c
src/mobility.c src/taplist_mtu.c src/filter.c src/general.c src/main.c
src/mcast.c src/protocol.c src/channel.c
INCLUDES = -I $(INCLUDE_NORMAL) -I $(INCLUDE_OPENSSL) -I
./include/hapd -I $(INCLUDE_HOSTAPD) -I$(INCLUDE_LIBNL) -I ./include
LIBRARIES = -L $(LIBRARY_NORMAL) -L $(LIBRARY_OPENSSL) -L $(LIB_DIR)
LDFLAGS = -lnl-3 -lnl-genl-3 -lssl -lcrypto

#export STAGING_DIR = $(OPENWRT_DIR)/staging_dir/
export STAGING_DIR=$(OPENWRT_DIR)/staging_dir/ $(CC) $(LIB_DIR)
$(INCLUDE_NORMAL)

EXECUTABLE = wifix_dualradio

# DO NOT CHANGE THIS

all: subsystem $(EXECUTABLE)

subsystem:
    $(MAKE) -C $(BASE_DIR)/hostap_x86/hostapd

hapd.o: src/hapd.c $(HEADERS_HOSTAPD)
    $(CC) $(CFLAGS) -c src/hapd.c $(INCLUDES)

$(EXECUTABLE): hapd.o $(SOURCE) $(DEPS)
    $(CC) $(CFLAGS) hapd.o $(SOURCE)
$(INCLUDE_HOSTAPD)/common/wpa_ctrl.o
$(INCLUDE_HOSTAPD)/utils/os_unix.o $(INCLUDES) $(LIBRARIES) -o $@
$(LDFLAGS)

.PHONY: clean

clean:
    rm -f $(EXECUTABLE) *.o
```

Appendix C.

Network configuration file

```
config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

config interface 'ctrl'
        option ifname 'eth0'
        option proto 'dhcp'

config interface 'wifi0'
        option proto  'static'
        option ipaddr '0.0.0.0'
        option netmask '0.0.0.0'

config interface 'wifi1'
        option proto 'static'
        option ipaddr '0.0.0.0'
        option netmask '0.0.0.0'
```

Appendix D

Wireless configuration file

```
config wifi-device 'radio0'
        option type 'mac80211'
        option path 'pci0000:00/0000:00:0c.0'
        option htmode 'HT20'
        option disabled '0'
        option log_level '1'
        option country 'PT'

config wifi-iface
        option device 'radio0'
        option network 'wifi0'
        option ssid 'MRMC MESH'
        option encryption 'none'
        option mode 'ap'

config wifi-device 'radio1'
        option type 'mac80211'

        option path 'pci0000:00/0000:00:0e.0'
        option htmode 'HT20'
        option disabled '0'
        option country 'PT'

config wifi-iface
        option device 'radio1'
        option network 'wifi1'
        option ssid 'MRMC MESH'
        option encryption 'none'
        option mode 'sta'
```

Appendix E.

This appendix contains the files that enable MAP's auto configuration.

Appendix E.1

This appendix contains the main.c file

```
int main(int argc, char *argv[])
{
    tap_list_info taplstinfo;
    mcast_group_list mc_gl;
    mobility_list moblist;
    char br_name[] = "br-lan";
    int sock_fd = 0, i = 0;
    char protocol[IFNAMSIZ] = {0};

    gotParent = FALSE;

    // ---------------------------- RESETs ---------------------------
-------
    //reset the interface and neighbour struct parameters
    for(i=0;i<2;i++)
        memset(&iface[i],0,sizeof(struct real_if));

    memset(&taplstinfo, 0, sizeof(tap_list_info)); //clears taplstinfo
    taplstinfo.tap_first = NULL;
    taplstinfo.tap_last = NULL;
    taplstinfo.tap_actual = NULL;

    memset(&mc_gl, 0, sizeof(mcast_group_list));
    mc_gl.mcast_group_first=NULL;
    mc_gl.mcast_group_last=NULL;
    mc_gl.mcast_group_actual=NULL;
    mc_gl.total_nr_mcast_groups=0;

    memset(&moblist, 0, sizeof(mobility_list));
    moblist.mobility_elem_first=NULL;
    moblist.mobility_elem_last=NULL;
    moblist.mobility_elem_actual=NULL;
    moblist.total_nr_mobility_elems=0;

    // ------------------------------ USAGE --------------------------
--------------
    if (argc != 6) {
        printf("USAGE: %s  <interface1> <interface2> <debug_level>
<mode> <version>\n<debug_level> = [0..3]\n<version> = [1 or
2]\n",argv[0]);
        exit(1);
    }
    //Debug Level
    if ((atoi(argv[3]) < 0) || (atoi(argv[3]) > 3 )) {
        printf("Usage: %s  <interface1> <interface2> <DEBUG LEVEL>
<mode> <version>\n<debug_level> = [0..3]\n<version> = [1 or
2]\n",argv[0]);
```

```
        exit(1);
    }
    //Version
    if((atoi(argv[5]) != 1) && (atoi(argv[5]) != 2)){
        printf("Usage: %s <debug_level> <master|slave>
<VERSION>\n<debug_level> = [0..3]\n<version> = [1 or 2]\n",argv[0]);
exit(1);
    }
    //Mode
    if(strcmp(argv[4], "master") == 0){
        strcpy(iface[0].name,"wlan0");
        gw = TRUE;
        ap_id = 0;
        puts("\n#### PREPARING MASTER NODE ####");
    }

    else{
        if(strcmp(argv[4], "slave") == 0)
            gw = FALSE;
        else{
            printf("Usage: %s <debug_level> <MASTER|SLAVE>
<version>\n<debug_level> = [0..3]\n<version> = [1 or 2]\n",argv[0]);
exit(1);
        }
    }

   // Wifix version parameters
    version = atoi(argv[5]); //yet only version 1 is supported
    debug_level = atoi(argv[3]);

    // ------------------------------------ INIT INTERFACES ---------
---------------------------------
    // Get interfaces and respective parameters
    if(getInterfaceParameters(argv[1],&iface[0]) == -1){
        printf("\n");
        exit(1);
    }
    if(strchr(iface[0].proto,'a') != NULL)
        iface[0].agg_band = 5;
    if(strchr(iface[0].proto,'b') != NULL)
        iface[0].agg_band += 2;
    printf("AGG_BAND = %d\n",iface->agg_band);

    iface_cnt++;

    if(!gw){
        if(getInterfaceParameters(argv[2],&iface[1]) == -1){
            printf("\n");
            exit(1);
        }
        iface_cnt++;
        if(strchr(iface[1].proto,'a') != NULL)
                iface[1].agg_band = 5;
            if(strchr(iface[0].proto,'b') != NULL)
                iface[1].agg_band += 2;
            printf("AGG_BAND = %d\n",iface->agg_band);
    }

    if(iface_cnt  <1){
        printf("No wireless Interfaces Detected...\nExiting...\n");
        exit(1);
```

```
    }
    printf("\n AGG_TOT(%d) = IF[0](%d) +
IF[1](%d)\n",(iface[0].agg_band +
iface[1].agg_band),iface[0].agg_band,iface[1].agg_band);
    if(gw && (iface[0].agg_band > 5))
        isfullDB = TRUE;

    if((iface_cnt > 1) && (!gw)){
        if((iface[0].agg_band + iface[1].agg_band) >= 14){
            isfullDB = TRUE;
            printf("\nBoth interfaces are Dual Band\n");
        }
    }
    for(i=0;i<iface_cnt;i++){
        printf("\nIFACE[%d]\n",i);
        printf("NAME: %s\n",iface[i].name);
        printf("PROTO: %s\n", iface[i].proto);
        printf("FD: %d\n",iface[i].fd);
        printf("-----------\n");
    }

    // ------------------------ Initialize system ------------------
---------------------
    for(i=0;i<iface_cnt;i++){
        // Bring interfaces from Dormant state
        printf("Setting %s UP: ",iface[i].name);
        if(setInterfaceState(iface[i].name,"up") == -1)
            printf("NOT OK\n");
        else printf("OK\n");

        // Set interface state
        if(gw){
            if(setIfaceMode(iface[i].name,"ap","11g",NULL,0,0,1) == 0
){//Set GW interface to AP mode
                if ((iface[i].fd = bind_dev(iface[i].name)) <0){
                    perror("no such device");
                    exit(1);
                }
            }
        }
        else{
            setIfaceMode(iface[i].name,"sta",NULL,NULL,0,0,0);//Set
MAP interfaces to STA mode
            if(i>0)

setIfaceMode(iface[i].name,"sta",NULL,NULL,0,0,1);//Set MAP interfaces
to STA mode
            /*
             * Don't bind to interface just yet as they should be
reinitialized once
             * a parent is found.
             */
        }
    }

    if(gw)
        taplstinfo.max_fd = iface[0].fd;

    // Estabelecida a ligação com o interface br_lan
    sock_fd = init(br_name);
```

```
    taplstinfo.max_fd = max(taplstinfo.max_fd, sock_fd);

    prot[0] = 0xFF;
    prot[1] = 0xFF;

    fflush(stdout);

    mainLoop(&taplstinfo, &mc_gl, &moblist, sock_fd, br_name);

    return 0;
```

Appendix E.2

This appendix contains the general.c file

```
#include <errno.h>
#include <fcntl.h>
#include <ifaddrs.h>
#include <linux/wireless.h>
//#include <net/if.h>
#include <netpacket/packet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#include "general.h"
#include "taplist.h"
#include "filter.h"
#include "bridgeutils.h"
#include "mcast.h"
#include "mobility.h"
#include "channel.h"

#define SSID "MRMC MESH"

void print_ifaces(real_if interface[]){
    int i=0;
    puts("\n-------");
    for(i=0;i<iface_cnt;i++){
        printf("Name: %s\n",interface[i].name);
        printf("Suported protocols: %s\n",interface[i].proto);
        printf("MAC addr:%.2x:%.2x:%.2x:%.2x:%.2x:%.2x\n",
interface[i].mac[0],interface[i].mac[1],interface[i].mac[2],
interface[i].mac[3], interface[i].mac[4],interface[i].mac[5]);
    }
}

int getChannelFromFeq(float frequency){
    // 2.4 GHz
    if(frequency >= 2412 && frequency < 2484)
        return (frequency - 2412) / 5 + 1;
    else if(frequency >= 2.412 && frequency < 2.484)
        return (frequency * 1000 - 2412) / 5 + 1;
    // 5 GHz
    else if(frequency >= 5170 && frequency <= 5825)
        return (frequency - 5170) / 5 + 34;
    else if(frequency >= 5.170 && frequency <= 5.825)
        return (frequency * 1000 - 5170) / 5 + 34;
    else return -1;
}

void malloc_error_handler(){
    printf("Unable to allocate memory for substr");
    exit(1);
```

```
}

//Opens raw socket  to real network interface.
//Only receives 0xFFFF headers
//Return value: opened interface file descriptor
int bind_dev(char *device)//Funciona
{
    struct sockaddr_ll addr;
    struct ifreq ifr;
    int if_fd = 0;

    if_fd = socket(PF_PACKET, SOCK_RAW, htons(0xffff));// if i wanted
to receive all frames..."htons(ETH_P_ALL)"
    if(debug_level != 0)
        printf("Binding to %s\n", device);
    memset(&ifr, 0, sizeof(ifr));
    strncpy(ifr.ifr_name, device, sizeof(ifr.ifr_name));

    if (ioctl(if_fd, SIOCGIFINDEX, &ifr) == -1) {
        perror("ioctl(SIOCGIFINDEX)");
        return -1;
    }

    // bind the packet socket to a specific interface
    memset(&addr, 0, sizeof(addr));
    addr.sll_family=AF_PACKET;
    addr.sll_protocol=htons(0xffff);
    addr.sll_ifindex=ifr.ifr_ifindex;
    addr.sll_hatype=0;
    addr.sll_pkttype=PACKET_OTHERHOST;
    addr.sll_halen=0;

    if ( bind(if_fd, (struct sockaddr *)&addr, sizeof(struct
sockaddr_ll)) ) {
        perror("bind()");
        return -1;
    }

    //changes tap MTU
    ifr.ifr_mtu = 1600;
    if (ioctl (if_fd, SIOCSIFMTU, &ifr) < 0)
    {
        perror("Tap MTU change:");
        return 0;
    }

    //Sets interface UP
    ifr.ifr_flags = IFF_UP;
    if (ioctl (if_fd, SIOCSIFFLAGS, &ifr) < 0)
    {
        perror("Interface NOT up:");
        return 0;
    }

    return if_fd;
}

int get_IP_Address(char * dev, unsigned char* ip_ad){

    struct ifreq ifr2;
    int fd2;
```

```
    int i;
    unsigned char ip_addr[4];
    unsigned int aux;

    fd2 = socket(AF_INET, SOCK_DGRAM, 0);
    ifr2.ifr_addr.sa_family = AF_INET;
    strncpy(ifr2.ifr_name, dev, IFNAMSIZ-1);
    ioctl(fd2, SIOCGIFADDR, &ifr2);
    close(fd2);

    aux = ((struct sockaddr_in *)&ifr2.ifr_addr)->sin_addr.s_addr;
    memcpy(ip_addr, &aux, sizeof(unsigned char)*4);

    // display result
    for(i = 0; i < 4; i++)
        printf("%u.",ip_addr[i]);
    printf("\n");

    memcpy(ip_ad, ip_addr, sizeof(unsigned char)*4);
    return 1;
}

int getMacAddress(char *ifname, unsigned char* mac_addr){//Funciona
    int fd;
    struct ifreq ifr;

    memset(&ifr, 0, sizeof(ifr));
    strncpy(ifr.ifr_name, ifname, IFNAMSIZ-1);

    ifr.ifr_addr.sa_family = AF_INET;

    if ((fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        fprintf(stderr,"SOCKET: %s\n",strerror(errno));
        return 0;
    }

    if (ioctl(fd, SIOCGIFHWADDR, &ifr) != -1) {
        close(fd);
        memcpy(mac_addr, ifr.ifr_hwaddr.sa_data, sizeof(unsigned
char)*6);//copies memory positions
                }
    return 1;
}

/*
 * Distiguish between wireless and non wireless interfaces
 */
int getWirelessProto(const char* ifname, char* protocol){
    int fd = -1;
    struct iwreq pwrq;

    memset(&pwrq, 0, sizeof(pwrq));
    strncpy(pwrq.ifr_name, ifname, IFNAMSIZ);

    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
        fprintf(stderr,"SOCKET: %s\n",strerror(errno));
        return -1;
    }

    if (ioctl(fd, SIOCGIWNAME, &pwrq) < 0) {
        fprintf(stderr,"ERROR(SIOCSIWMODE): %s: ",strerror(errno));
```

```
        close(fd);
        return -1;
    }

     if (protocol)
         strncpy(protocol, pwrq.u.name, IFNAMSIZ);

    close(fd);
    return 0;
}

void show_frame(char *message ,unsigned char *frame_buf, int
frame_bufsize){

    int i=0;
    printf("\n  --------| %s |--------\n",message);
    printf("  Dst:      %02x:%02x:%02x:%02x:%02x:%02x\n",
frame_buf[0],frame_buf[1],frame_buf[2],frame_buf[3],frame_buf[4],frame
_buf[5]);
    printf("  Src:      %02x:%02x:%02x:%02x:%02x:%02x\n",
frame_buf[6],frame_buf[7],frame_buf[8],frame_buf[9],frame_buf[10],fram
e_buf[11]);
    printf("  Protocol: %02x%02x\n", frame_buf[12],frame_buf[13]);

    printf("  Payload:\n  ");
    for(i = 14; i<frame_bufsize; i++)
    {
        printf("%02x ", frame_buf[i]);
        if((i-13)%16 == 0){
            printf("\n  ");
        }else{
            if((i-13)%8 == 0)
                printf("  ");
        }
    }
    printf ("\n  Total:%d bytes in frame", frame_bufsize);
    printf("\n\n");
}

int getInterfaceParameters(char *ifname, real_if *interface){
    int i = 0, ret = -1;
    char protocol[IFNAMSIZ] = {0};

    if(getWirelessProto(ifname,protocol) > -1){
        memcpy(interface->name,ifname,sizeof(interface->name));
        memcpy(interface->proto,protocol,sizeof((protocol)));
        printf("interface %s is wireless: %s \n", interface->name,
interface->proto);
        getMacAddress(ifname,interface->mac);
        ret = 0;
    }

    return ret;
}

int setInterfaceState(const char *ifname, char *state){
    struct ifreq ifr;
    int fd = -1;

    memset(&ifr,0,sizeof(ifr));
    strncpy(ifr.ifr_name, ifname, IFNAMSIZ);
```

```
    if((fd = socket(AF_INET,SOCK_STREAM,0))==-1){
        perror("Socket");
        return -1;
    }

    ifr.ifr_flags |= IFF_UP;

    if(ioctl(fd,SIOCSIFFLAGS,&ifr)< 0){
        fprintf(stderr,"ERROR(SIOCSIWMODE): %s\n",strerror(errno));
        close(fd);
        return -1;
    }
    close(fd);
    return 0;
}

//Definir o modo de operação da interface AP/STA
int setIfaceMode(char *ifname, char *mode, char *hwmode, int
up_chnls[], int parent_chnl, int hops, int wl_restart){
    char cmd[512];
    char buf[256];
    int niface;
    FILE *fp;

    extern FILE *popen();


    if(strcmp(ifname,"wlan0")==0)
        niface = 0;
    else
        niface = 1;

    // Interfaces Operating in AP might be assigned a channel
    if(strcmp(mode,"ap") == 0){
        sprintf(cmd,"uci set wireless.radio%d.hwmode=%s; "
                "uci set wireless.@wifi-iface[%d].mode=%s; "
                "uci set wireless.@wifi-
iface[%d].ssid=\"%s\";",niface,hwmode,niface,mode,niface,SSID);

        // Assign a channel if it's not the GW
        if(!gw){
            int channel = 0;
            // Choose best channel
            channel = choose_channel(up_chnls,parent_chnl,hops);
            sprintf(cmd,"%s uci set
wireless.radio%d.channel=%d;",cmd,niface,channel);
        }
    }
    // Set interface to STA mode - Channel is assigned by the parent
    else {
        sprintf(cmd,"uci set wireless.@wifi-
iface[%d].mode=%s;",niface,mode);
    }
    if(wl_restart)
        sprintf(cmd,"%s wifi",cmd);


    while(!(fp = popen(cmd,"r"))){
        exit(1);
    }
```

```
    pclose(fp);

    //Wait for all Wireless operations to be completed
    if(wl_restart)
        sleep(5);

    //------------------- Validate assignment -----------------
    memset(&cmd,0,sizeof(cmd));
    fflush(stdout);

    sprintf(cmd,"uci show wireless | grep mode=%s",mode);

    while(!(fp = popen(cmd,"r"))){
        exit(1);
    }
    while(fgets(buf,sizeof(buf),fp)){
        if(niface == 0){

            if((strncmp(buf+10,"wifi-iface[0]",3)!=0) ||
(strncmp(buf+10,"wifi-iface[1]",3)!=0)){
                printf("DEBUG BUFFER::::%s\n",buf);
                printf("DEBUG BUFFER+10::::%s\n",buf+10);
                pclose(fp);
                return -1;
            }
        }
    }
    pclose(fp);
    fflush(stdout);
    return 0;
}

int check_max_fd(tap_list_info *taps_info, real_if *interface, int
sock_fd){//Funciona

    int max_fd=0;
    taps_info->tap_actual = taps_info->tap_first;
    while (taps_info->tap_actual != NULL){
        max_fd = max(taps_info->tap_actual->fd, max_fd);
        taps_info->tap_actual = taps_info->tap_actual->next_tap;
    }
    max_fd = max(interface->fd, max_fd);
    max_fd = max(sock_fd, max_fd);
    return max_fd;
}

int init(char *bridge_name)
{
    int skfd = 0;

    br_init();
    br_cmd_addbr(bridge_name);
    br_cmd_setfd(bridge_name);
    br_cmd_stp(bridge_name);
    br_cmd_setageing(bridge_name);
    skfd = setCtrlSocket();

    return skfd;
}
```

```
int file_top_write(real_if *iface){

    int file_fd = 0, i = 0;
    char buf[100];
    char start[] = {"digraph G {\n"};
    char end[] = {"}\n"};
    printf("Top_list_count => %d\tEscrita fich\n", top_list_cnt);
    if( (file_fd = open("/tmp/topology.dot", O_CREAT | O_TRUNC |
O_NONBLOCK | O_WRONLY, S_IRWXU | S_IXGRP | S_IXOTH)) == -1){
        perror("TOPOLOGY file");
        return 1;
    }
    if(top_list_cnt == 0){
        close(file_fd);
        return 0;
    }

    write(file_fd, start, 14);
    sprintf(buf, "\t\"%02X%02X%02X%02X%02X%02X\" [shape=box];\n",
iface->mac[0], iface->mac[1],iface->mac[2],iface->mac[3],iface-
>mac[4], iface->mac[5]);
 write(file_fd, buf, 29);

    for(i=0; i<top_list_cnt; i++){
        sprintf(buf, "\t\"%02X%02X%02X%02X%02X%02X\" ->
\"%02X%02X%02X%02X%02X%02X\";\n", top_list[i].node_mac[0],
                        top_list[i].nb_mac[1],
                        top_list[i].nb_mac[2],
                        top_list[i].nb_mac[3],
                        top_list[i].nb_mac[4],
                        top_list[i].nb_mac[5],
                        top_list[i].node_mac[0],
                        top_list[i].node_mac[1],
                        top_list[i].node_mac[2],
                        top_list[i].node_mac[3],
                        top_list[i].node_mac[4],
                        top_list[i].node_mac[5]);
        write(file_fd, buf, 35);
    }

    write(file_fd, end, 2);

    close(file_fd);
    return 0;
}

int write_tap_on_file(tap_node *tap, char * msg)
{
    group_list_file = fopen("group_list.txt", "a");
    fprintf(group_list_file, "%sNr %d - Name %s - FD %d - DestMAC
%.2x:%.2x:%.2x:%.2x:%.2x:%.2x - TTL %d - Up Neighbour Flag %d\n",msg
,tap->nr, tap->name, tap->fd,
            tap->mac[0],
            tap->mac[1],
            tap->mac[2],
            tap->mac[3],
            tap->mac[4],
            tap->mac[5],
            tap->ttl,
            tap->up_nb_flag);
    fclose(group_list_file);
```

```
    return 0;
}

int write_taplist_on_file(int totalnrtaps, int maxfd, char * msg)
{
    group_list_file = fopen("group_list.txt", "a");
    if(totalnrtaps != 0)
        fprintf(group_list_file, "%s %d\n\n", msg, totalnrtaps);
    else{
        if(maxfd != 0)
            fprintf(group_list_file, "%s %d\n", msg, maxfd);
        else
            fprintf(group_list_file, "%s", msg);
    }
    fclose(group_list_file);
    return 0;
}
```

Appendix E.3

This appendix contains the mainloop.c file

```
#include <errno.h>
#include <linux/nl80211.h>
#include <net/if.h>
#include <netinet/ether.h>
#include <netlink/netlink.h>
#include <netlink/socket.h>
#include <netlink/msg.h>
#include <netlink/genl/genl.h>
#include <netlink/genl/family.h>
#include <netlink/genl/ctrl.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

#include "bcast.h"
#include "filter.h"
#include "forward.h"
#include "general.h"
#include "hapd.h"
#include "mainloop.h"
#include "mcast.h"
#include "mobility.h"
#include "protocol.h"
#include "taplist.h"

neighbour parent;

static int no_seq_check(struct nl_msg *msg, void *arg){
    return NL_OK;
}

static int error_handler(struct sockaddr_nl *nla, struct nlmsgerr
*err, void *arg) {
    // Callback for errors.
    printf("error_handler() called.\n");
    int *ret = arg;
    *ret = err->error;
    return NL_STOP;
}

static int ack_handler(struct nl_msg *msg, void *arg) {
    // Callback for NL_CB_ACK.
    int *ret = arg;
    *ret = 0;
    return NL_STOP;
}

static int family_handler(struct nl_msg *msg, void *arg) {

    struct handler_args *grp = arg;
    struct nlattr *tb[CTRL_ATTR_MAX + 1];
    struct genlmsghdr *gnlh = nlmsg_data(nlmsg_hdr(msg));
    struct nlattr *mcgrp;
```

```
    int rem_mcgrp;

    nla_parse(tb, CTRL_ATTR_MAX, genlmsg_attrdata(gnlh, 0),
genlmsg_attrlen(gnlh, 0), NULL);

    if (!tb[CTRL_ATTR_MCAST_GROUPS]) return NL_SKIP;

    nla_for_each_nested(mcgrp, tb[CTRL_ATTR_MCAST_GROUPS], rem_mcgrp)
{  // This is a loop.
        struct nlattr *tb_mcgrp[CTRL_ATTR_MCAST_GRP_MAX + 1];

        nla_parse(tb_mcgrp, CTRL_ATTR_MCAST_GRP_MAX, nla_data(mcgrp),
nla_len(mcgrp), NULL);

        if (!tb_mcgrp[CTRL_ATTR_MCAST_GRP_NAME] ||
!tb_mcgrp[CTRL_ATTR_MCAST_GRP_ID]) continue;
        if (strncmp(nla_data(tb_mcgrp[CTRL_ATTR_MCAST_GRP_NAME]), grp-
>group,
                nla_len(tb_mcgrp[CTRL_ATTR_MCAST_GRP_NAME]))) {
            continue;
        }

        grp->id = nla_get_u32(tb_mcgrp[CTRL_ATTR_MCAST_GRP_ID]);
        break;
    }

    return NL_SKIP;
}

/*
 * Formata o campo do endereço MAC recebido no formato NL_ATTR_MAC
 * para uma string.
 * mac_addr_n2a(macbuf, nla_data(tb[NL80211_ATTR_MAC]));
 */
void mac_addr_n2a(char *mac_addr, unsigned char *arg) {
    int i, l;

    l = 0;
    for (i = 0; i < 6; i++) {
        // Primeira iteração não adiciona ":" à esquerda
        if (i == 0) {
            sprintf(mac_addr+l, "%02x", arg[i]);
            l += 2;
        } else { // nas restantes interações adiciona ":" à esquerda
            sprintf(mac_addr+l, ":%02x", arg[i]);
            l += 3;
        }
    }
}

int nl_get_multicast_id(struct nl_sock *sock, const char *family,
const char *group) {
    struct nl_msg *msg;
    struct nl_cb *cb;
    int ret, ctrlid;
    struct handler_args grp = { .group = group, .id = -ENOENT, };

    msg = nlmsg_alloc();
    if (!msg)
        return -ENOMEM;
```

```
    cb = nl_cb_alloc(NL_CB_DEFAULT);
    if (!cb) {
        ret = -ENOMEM;
        goto out_fail_cb;
    }

    ctrlid = genl_ctrl_resolve(sock, "nlctrl");

    genlmsg_put(msg, 0, 0, ctrlid, 0, 0, CTRL_CMD_GETFAMILY, 0);

    ret = -ENOBUFS;
    NLA_PUT_STRING(msg, CTRL_ATTR_FAMILY_NAME, family);

    ret = nl_send_auto_complete(sock, msg);
    if (ret < 0)
        goto out;

    ret = 1;

    nl_cb_err(cb, NL_CB_CUSTOM, error_handler, &ret);
    nl_cb_set(cb, NL_CB_ACK, NL_CB_CUSTOM, ack_handler, &ret);
    nl_cb_set(cb, NL_CB_VALID, NL_CB_CUSTOM, family_handler, &grp);

    while (ret > 0)
        nl_recvmsgs(sock, cb);

    if (ret == 0)
        ret = grp.id;

    nla_put_failure:
        out:
            nl_cb_put(cb);
        out_fail_cb:
            nlmsg_free(msg);
    return ret;
}

static int prepare_listen_events(struct sock_state *state){
    int mcid, ret;

    // Subscibe to MLME events
    mcid = nl_get_multicast_id(state->nl_sock, "nl80211", "mlme");
    if(mcid >= 0){
        printf("\nAdded MCAST ID: %d",mcid);
        ret = nl_socket_add_membership(state->nl_sock,mcid);
        if(ret)
            return ret;
    }
    return 0;
}

/*
 * Socket Creation and Binding
 * On success returns the socket file descriptor. On error returns
 * errno.
 */
static int nl_init(struct sock_state *state, int nlproto){

    state->nl_sock = nl_socket_alloc();

    if(!state->nl_sock){
```

```
        fprintf(stderr,"\nFailed to allocate netlink socket");
        return errno;
    }

    nl_socket_set_buffer_size(state->nl_sock, 8192, 8192);

    if(nl_connect(state->nl_sock,nlproto) < 0 ){
        fprintf(stderr,"\nFailed to connect: %s
(%d)",strerror(errno),errno);
        return errno;
    }

    return nl_socket_get_fd(state->nl_sock);
}



int mlme_event(struct nl_msg *msg, void *arg){
    // Callback function for events related to the GENERIC netlink
socket
    struct genlmsghdr *gnlh = nlmsg_data(nlmsg_hdr(msg));
    struct print_event_args *args = arg;
    struct nlattr *tb[NL80211_ATTR_MAX +1];
    char ifname [100];
    char macbuf[6*3];

    nla_parse(tb, NL80211_ATTR_MAX, genlmsg_attrdata(gnlh, 0),
genlmsg_attrlen(gnlh, 0), NULL);

    if (tb[NL80211_ATTR_IFINDEX] && tb[NL80211_ATTR_WIPHY]) {
        if_indextoname(nla_get_u32(tb[NL80211_ATTR_IFINDEX]), ifname);
        printf("\n%s (phy #%d): ", ifname,
nla_get_u32(tb[NL80211_ATTR_WIPHY]));
    }

    switch (gnlh->cmd){

    case NL80211_CMD_NEW_STATION:
        mac_addr_n2a(macbuf, nla_data(tb[NL80211_ATTR_MAC]));
        printf("new station %s\n", macbuf);
        addDownNeighbour(args->taplistp,
nla_data(tb[NL80211_ATTR_MAC]), args->bridge, &iface[ap_id], args-
>parentp);
        break;

    case NL80211_CMD_DEL_STATION:
        mac_addr_n2a(macbuf, nla_data(tb[NL80211_ATTR_MAC]));
        printf("\ndel station %s\n", macbuf);
        checkOldDownNeighbour(args-
>taplistp,nla_data(tb[NL80211_ATTR_MAC]),args->bridge, &iface[ap_id]);
        break;

    case NL80211_CMD_DISCONNECT:
        printf("disconnected:");
        if(tb[NL80211_ATTR_DISCONNECTED_BY_AP])
            printf("by AP\n");
        else
            printf(" (local request)");
            printf("\n");
        break;
    }
```

```
    return 0;
}

int find_parent(){
    int i, nb_num = -1, scan_complete = FALSE;

    printf("\n############ BEACON SCAN ############\n");
    while((nb_num < 0) && (scan_complete == FALSE)){        // Lock
until a possible neighbor is detected
        if(isfullDB){       // Both interfaces support dual-band
            printf("\n#Scanning ONLY on interface
%s\n",iface[0].name);
            if((nb_num = beacon_scan(iface[0].name)) > -2)
                scan_complete = TRUE;
        }
        else{
            printf("\n#Scanning on interface %s\n",iface[0].name);
            if((nb_num = beacon_scan(iface[0].name)) > -2){
                printf("\n#Scanning on interface %s\n",iface[1].name);
                if((nb_num += beacon_scan(iface[1].name)) > -2)
                    scan_complete = TRUE;
            }
    }

    if(nb_num > -1)
        print_neighbors(nb,nb_num);

    if(choose_neighbor(nb_num, &parent) == 0){
        // Verificar o canal utilizado pela estação Parent
        if(parent.freq > 2500){     //If parent is 802.11a
            printf("Parent freq MAIOR que 2500 A\n");
            if(isfullDB || ((iface[0].agg_band >= 5) &&
(iface[1].agg_band != 5 ))){
                ap_id = 0; sta_id = 1;
            }
            else{
                ap_id = 1; sta_id = 0;
            }

if(setIfaceMode(iface[ap_id].name,"ap","11g",parent.tree_chan_lst,getC
hannelFromFeq(parent.freq),parent.hops_to_gw+1,0)==0){

if(setIfaceMode(iface[sta_id].name,"sta",NULL,parent.tree_chan_lst,get
ChannelFromFeq(parent.freq),parent.hops_to_gw+1,1)==0){

                    strcpy(ap_iface,iface[ap_id].name);
                    if(isfullDB){
                        printf("\nBoth Interfaces have 802.11abgn
capabilities\n");
                        printf("\nConnecting to parent using interface
wlan0\n");
                        connect_parent(&parent);
                    }
                    else{   //Use the default 5GHz interface (wlan0)
                        printf("\nConnecting to parent using DEFAULT
interface wlan0\n");
                        connect_parent(&parent);
                    }
                }
            }
```

```
            else fprintf(stderr,"returned other than zero\n");
        }

        else{       //If parent is 802.11bg
            printf("Parent freq MENOR que 2500 BG\n");
            if(isfullDB || ((iface[0].agg_band != 5) &&
(iface[1].agg_band >= 5 ))){
                ap_id = 1; sta_id = 0;
            }
            else {
                ap_id = 0; sta_id = 1;
            }

if(setIfaceMode(iface[ap_id].name,"ap","11a",parent.tree_chan_lst,getC
hannelFromFeq(parent.freq),parent.hops_to_gw+1,0)==0){

if(setIfaceMode(iface[sta_id].name,"sta","11g",parent.tree_chan_lst,ge
tChannelFromFeq(parent.freq),parent.hops_to_gw+1,1)==0){
                    strcpy(ap_iface,"wlan0");
                    if((isfullDB)){
                        printf("\nBoth Interfaces have 802.11abgn
capabilities");
                        printf("\nConnecting to parent using interface
wlan1");
                        connect_parent(&parent);
                    }
                    else if((strncmp("bg",iface[0].proto,2) > 0)){
//Use the default 2.4GHz interface (wlan1)
                        printf("\nConnecting to parent using DEFAULT
interface %s",iface[1].name);
                        connect_parent(&parent);
                    }
                }
            }
            else fprintf(stderr,"returned other than zero\n");
        }
    }
    fflush(stdout);
    return nb_num;
}

void mainLoop(tap_list_info *tli, mcast_group_list *mgl, mobility_list
*mbl, int ctrl_sock_fd, char brname[])
{
    int select_value = 0, frame_size = 0, timeout = 0, choice = 0,
iface_selected;
    int i = 0, j = 0, k, z = 0, dbg = 0, DHCP_request_from_nb = 0,
nb_cnt = -1, fd_gen, fd_rt, err = 0;
    int match = FALSE, difer = FALSE;
    //int broadcastOverTunnels = 1;
    unsigned char frame[1600]; // Dev MTU = 1600
    unsigned char bcast[] = {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};
    unsigned char mcast[] = {0x00};
    unsigned char mc_to_bc_addr1[] = {0xE0, 0x00, 0x00, 0x01};
    unsigned char mc_to_bc_addr2[]= {0xE0, 0x00, 0x00, 0x16};
    unsigned char mc_addr_aux[4];
    unsigned char mac_actual_aux[6];
    unsigned char ethtypeIP[] = {0x08, 0x00};
    unsigned char ptclUDP[] = {0x11};
    unsigned char portDHCP[] = {0x00, 0x43, 0x00, 0x44};
    unsigned char msgtypeRequest[] = {0x03};
```

```
    unsigned char msgtypeAck[] = {0x05};
    unsigned char transID[4];
    unsigned char srcMAC[6];
    unsigned char IGMP_type1[] = {0x16};
    unsigned char IGMP_type2[] = {0x17};
    unsigned char ptcl_IGMP[] = {0x02};
    unsigned char trans_id_list[10][4];
    unsigned char zero[] = {0x00, 0x00, 0x00, 0x00};
    unsigned char node_IP[4];

    int send_IGMP_query = 0;

    fd_set fds;
    struct timeval ttl_taps, t_prev, t_sel;
    struct sock_state nlstate, rtstate;
    struct print_event_args args;
    struct nl_cb *cb = nl_cb_alloc(NL_CB_DEFAULT);
    struct sockaddr_nl addr_nl;
    struct nl_sock *nls;
    neighbour nb;

    get_IP_Address("br-lan", node_IP);

    //Pode dar problemas!
    mcast_group *mcgr = NULL;

    mobility_elem *mob_elem_rem = NULL;

    for(k=0; k<10; k++){
        memcpy(trans_id_list[k], zero, 4*sizeof(unsigned char));
    }

    //INITIALIZES HASH LIST
/*************************************************************************
*****************************************/
    hash_lst.hash_list_size = 0;
    hash_lst.hash_actual = NULL;
    hash_lst.hash_first = NULL;
    hash_lst.hash_last = NULL;

/*************************************************************************
    // Control interface socket
    if(listen(ctrl_sock_fd, 1) == -1){
        perror("Listen");
        exit(1);
    }

    memset(&nb, 0, sizeof(neighbour));
    nb.up_nb = FALSE;
    //initializes select time out
    gettimeofday(&t_prev, NULL);
    t_sel.tv_sec = 0;
    t_sel.tv_usec = 150000;

    if(gw){
        ap_id = 0;
        printf("\nIFACE MASTER MODE: %s ",iface[ap_id].name);
        if(set_beacon_ie221(iface[ap_id].name,0,0,0)!=0){
            perror("Beacon_set");
        }
    }
```

```
    if(!gw && !gotParent){
        if((nb_cnt = find_parent()) >= 0 ){
            if(set_beacon_ie221(iface[ap_id].name,
parent.hops_to_gw+1, parent.tree_chan_lst,
getChannelFromFeq(parent.freq))==0){
            // Inserir uma nova TAP para o pai à lista de TAPs
                for(i=0;i<iface_cnt;i++){
                    if ((iface[i].fd = bind_dev(iface[i].name)) < 0){
                        perror("no such device");
                        exit(1);
                    }
                tli->max_fd = max(iface[i].fd,tli->max_fd);
                }
            insert_tap_node(tli, parent.nb_mac, &iface[sta_id],
brname, 1);
            }
        }
    }

    args.taplistp = tli;
    args.bridge = brname;
    args.parentp = &parent;

    // -------------- NETLINK INITIALIZATION ---------------
    if (!cb) {
        fprintf(stderr, "failed to allocate netlink callbacks\n");
    }

    //------------ Inicialização do socket GENERIC
    fd_gen = nl_init(&nlstate, NETLINK_GENERIC);
    if(fd_gen < 0)
        printf("ERROR: Couldn't initialize Generic netlink socket");

    err = prepare_listen_events(&nlstate);

    //Set Callback functions
    nl_cb_set(cb, NL_CB_SEQ_CHECK, NL_CB_CUSTOM, no_seq_check, NULL);
    nl_cb_set(cb, NL_CB_VALID, NL_CB_CUSTOM, mlme_event, &args);
    for(i=0;i<iface_cnt;i++)
        printf("\n—DEBUG:NAME:%s\nFD:%d\n",iface[i].name,iface[i].fd);

    while(1){
        // If its not the Gateway

        //prepares fd set
        FD_ZERO(&fds);
        // Adiciona o descritor da interface real 0 à lista de FDs
        FD_SET(iface[0].fd, &fds);
        // Adiciona o descritor da interface real 1 à lista de FDs
        if(!gw)
            FD_SET(iface[1].fd, &fds);
        // Adiciona o descritor da interface de controlo à lista de
interfaces
        FD_SET(ctrl_sock_fd, &fds);
        //adds taps to select set
        tli->tap_actual=tli->tap_first;
        while (tli->tap_actual != NULL){
            FD_SET(tli->tap_actual->fd, &fds);
            tli->tap_actual = tli->tap_actual->next_tap;
        }
```

```
        // Add the Generic family socket to the FD set
        FD_SET(fd_gen,&fds);

        //listens on fds
        select_value = 0;

        if((select_value = select(tli->max_fd+4, &fds, NULL, NULL,
&t_sel)) < 0){
            fprintf(stderr,"Error on select:%s\n",strerror(errno));
        }

        else{
            // Verifica se o bit do FD da interface br_lan foi ativo
            if (FD_ISSET(ctrl_sock_fd, &fds)){
                get_cmd(20, ctrl_sock_fd, iface, &nb);
            }

            if(select_value == 0){
                t_sel.tv_sec = 0;
                t_sel.tv_usec = 250000;
            }
            else{
                tli->tap_actual = tli->tap_first;//points to beginning
of list
                //
================================================================================
                // checks if frame was received on real interface
================================================================================
                if ((FD_ISSET(iface[0].fd, &fds)) ||
(FD_ISSET(iface[1].fd, &fds))){
                    //printf("received on REAL IFACE \n");
                    // Activates a flag for the FD of the selected
interface
                    if(FD_ISSET(iface[0].fd, &fds))
                        iface_selected = 0;
                    else iface_selected = 1;
                    //printf("received on REAL IFACE
%s\n",iface[iface_selected].name);
                    //frame size - tamanho em bytes
                    if((frame_size = read(iface[iface_selected].fd,
frame, sizeof(frame)))<=0){
                        show_frame("* ERROR ON READ from iface",
frame, frame_size);
                    }

                    if(frame_size > 0){
                        if(send_IGMP_query == 1)
                            send_IGMP_query++;
                        if(debug_level >= 1 )
                            printf("frame(%d) received on REAL IFACE
%s\n",frame_size,iface[iface_selected].name);

                        //BROADCAST PROCESSING
------------------------------------------------------------------
                        if( (memcmp(frame, bcast, sizeof(bcast)) == 0)
&& (memcmp(frame+14, bcast, sizeof(bcast)) == 0)
                                && (memcmp(frame+26, prot,
2*sizeof(unsigned char)) != 0) ){

                            if(debug_level >= 2){
```

```
                                //puts("Broadcast frame received from
interface");
                                show_frame("Bcast received from
iface", frame, frame_size);
                            }
                            broadcastFrameFromIface(tli, frame,
frame_size);
                        }else{
                        // UNICAST PROCESSING
                            if(debug_level >= 2)
                                puts("Frame received from interface");
                            fwd_packet(tli, frame, frame_size);
                        }
                    }
                }

                //
================================================================
                // checks if frame was received on one of the taps
================================================================
flag_first = TRUE;
                while (tli->tap_actual != NULL){
                    if (FD_ISSET(tli->tap_actual->fd, &fds)) {
                        frame_size = read(tli->tap_actual->fd, frame,
sizeof(frame));
                        if(frame_size < 0 )
                            fprintf(stderr,"ERROR ON READ FROM %s: %s
\n",tli->tap_actual->name,strerror(errno));

                        //---debug---
                        else{
                            if(debug_level >= 2)
                                printf("frame(%d) received from %s\n
on FD:%d",frame_size,tli->tap_actual->name,tli->tap_actual->fd);
                            if(debug_level == 3)
                                show_frame("frame received from tap",
frame, frame_size);
                        }
                        //---
                        //BROADCAST PROCESSING
                        memcpy(mcast, frame, sizeof(unsigned
char));//verificar se o MAC de destino é multicast
                        if(((mcast[0] % 2) != 0) || mcast[0] == 1){
                            // Check if DST is the bcast addr
check the frame protocol type
                            if((memcmp(frame, bcast, sizeof(bcast)) ==
0) && (memcmp(frame+12, prot, 2*sizeof(unsigned char)) != 0)){
                                if(debug_level >= 2){
                                    show_frame("Bcast received from
tap", frame, frame_size);
                                }
                                broadcastFrameFromTap(tli, frame,
frame_size, parent.nb_mac);
                            }
                            // Multicast frame received
                            else{
                                if(debug_level >= 2){
                                    show_frame("Mcast received from
tap", frame, frame_size);
                                }
```

```
                                broadcastFrameFromTap(tli, frame,
frame_size, parent.nb_mac);
                            }
                        }else{
                            //UNICAST PROCESSING
                            if(debug_level >= 1)
                                printf("\n------- UNICAST FROM TAP----
--\n");
                            if(memcmp(frame+12, ethtypeIP,
2*sizeof(unsigned char)) == 0){ //IPv4?
                                if(memcmp(frame+23, ptclUDP,
sizeof(unsigned char)) == 0){ //UDP?
                                    if(memcmp(frame+34, portDHCP+2,
2*sizeof(unsigned char)) == 0){ //DHCP Request?
                                        if(memcmp(frame+284,
msgtypeRequest, sizeof(unsigned char)) == 0){
                                            if(debug_level > 0){
                                                printf("\nDHCP Request
Recebido na TAP: %.2x\n", frame[284]);

write_taplist_on_file(0, 0, "\n========== DHCP Request received
============\n");
                                            }
                                            memcpy(srcMAC, frame+6,
6*sizeof(unsigned char));
                                            memcpy(transID, frame+46,
4*sizeof(unsigned char));

//write_trans_id_list_on_file(trans_id_list);
                                            for(k=0; k<10; k++){
                                                if(transID[0] ==
trans_id_list[k][0] && transID[1] == trans_id_list[k][1] && transID[2]
== trans_id_list[k][2] && transID[3] == trans_id_list[k][3]){

memcpy(trans_id_list[k], zero, 4*sizeof(unsigned char));

DHCP_request_from_nb = 1;
                                                }
                                            }
                                        if(DHCP_request_from_nb == 0)

insert_mobility_elem(srcMAC, transID, mbl);
                                            else
                                          DHCP_request_from_nb = 0;

show_mobility_list_info(mbl);

//write_trans_id_list_on_file(trans_id_list);
                                        }
                                    }
                                }
                            }
                            //
                            // Check up_nb_flag ---- is up_nb ---->
iface[sta_id]
                            //                   |
                            //                   |
                            //                   ---- not up_nb --->
iface[ap_id]
                            //
                            if(tli->tap_actual->up_nb_flag == 1){
```

```
                                process_send(iface[sta_id].fd,
iface[sta_id].mac, tli->tap_actual->mac, frame, frame_size);
                            }
                            else{
                                process_send(iface[ap_id].fd,
iface[ap_id].mac, tli->tap_actual->mac, frame, frame_size);
                            }
                        }
                    }
                    tli->tap_actual = tli->tap_actual->next_tap;
                }
                // Verifica se foi ativo o bit dos FD correspondentes
aos sockets netlink
                if(FD_ISSET(fd_gen,&fds)){
                    nl_recvmsgs(nlstate.nl_sock, cb);
                }
                else if(FD_ISSET(fd_rt,&fds)){
                    //fprintf(stderr,"\nEvent Received:\n");
                    if(!gw)
                        read_event(fd_rt);
                }
            }
            // If none of the descriptors was ready to be written

            gettimeofday(&ttl_taps, NULL);

            if(ttl_taps.tv_sec-t_prev.tv_sec > 1){
                //---debug---
                if(debug_level == 3)
                    printf("select TIME OUT\n");
                //---
                timeout++;

                tli->tap_actual = tli->tap_first;
                while (tli->tap_actual != NULL) {
                    if(tli->tap_actual->up_nb_flag == 0);
                    //tli->tap_actual->ttl--;           ALTERADO -
DUALRADIO
                    tli->tap_actual = tli->tap_actual->next_tap;
                }

                //---debug---
                if(debug_level > 1){

printf("\n*************************************\n");
                    write_taplist_on_file(0, 0,
"\n*************************************\n");
                    show_tap_list_info(tli);

printf("\n*************************************\n");
                    write_taplist_on_file(0, 0,
"\n*************************************\n");
                }
                if(debug_level == 1){
                    puts("\e[2J");
                    show_tap_list_info(tli);
                }
                //---
                tli->tap_actual = tli->tap_first;
                while ( tli->tap_actual != NULL ) {
```

```
                    if (tli->tap_actual->ttl == 0){ // Enviar
Desassociação???
                        remove_tap_node(tli, tli->tap_actual, iface,
brname, ctrl_sock_fd);
                    }
                    else
                        tli->tap_actual = tli->tap_actual->next_tap;
                }

                // REMOVES ALL HASH LIST ENTRIES WICH EXPIRED
                hash_lst.hash_actual = hash_lst.hash_first;
                while(hash_lst.hash_actual != NULL){
                    if( (ttl_taps.tv_sec - hash_lst.hash_actual-
>time_el.tv_sec) > 2){
                        if(debug_level >= 2)
                            puts("Timer do elemento hash expirou.");
                        remove_hash_entry(hash_lst.hash_actual);
                    }else
                        hash_lst.hash_actual = hash_lst.hash_actual-
>next;
                }

                if( (timeout == 6) && (gw == FALSE) ){
                    //puts("\n\t\t\tCHOOSING NEIGHBOUR\n");


                }
                if(timeout > 6)
                    timeout = 0;

                gettimeofday(&t_prev, NULL);
            }

            //MODIFICADO WIFI-DUALRADIO
        }
        fflush(stdout);
    }
}
```

Appendix .E4

This appendix contains the protocol.c file

```
// <filename>.c
// <description>
// <version> (<date>)

// local includes
#include "protocol.h"
#include "general.h"
// system includes
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/un.h>
#include <linux/if_ether.h> /* out of habit, I use a SOCK_PACKET
socket */
#include <netinet/in.h>

// local prototypes

// exported methods implementation

void print_neighbors(neighbour nb[], int n){
    int i = 0, j = 0;
    puts("\n-------\n");
    for(i=0;i<=n;i++){
        printf("NB MAC:
%.2x:%.2x:%.2x:%.2x:%.2x:%.2x\n",nb[i].nb_mac[0],nb[i].nb_mac[1],nb[i]
.nb_mac[2],nb[i].nb_mac[3],nb[i].nb_mac[4],nb[i].nb_mac[5]);
        printf("FREQ: %d\n",nb[i].freq);
        printf("HOPS: %d\n",nb[i].hops_to_gw);
        for(j=0;j<nb[i].hops_to_gw;j++)
            printf("Channl LST: %d\n",nb[i].tree_chan_lst[j]);
        puts("------------\n");
    }
}

int beacon_scan(char *iface){

    struct nb_aux{
        char nb_mac[18];
        int freq;
        char hops_to_gw[3];
        char up_nb_mac[18];
        char data[30];
    };

    char *pch;
    char cmd[25], buf[BUFSIZ], mac_aux[18], delim[] = " ";
    int i = 0, type = 0, nb_counter = -1, cnt = 0;
    struct nb_aux nbs_a[10];

    //int freq = 0;
    int frqaux = 0, bss_cnt = 0, line_cnt = 0;

    FILE *fp;
```

```
    extern FILE *popen();
    //debug--
    //printf("\niw dev %s scan -u\n",iface);
    //--
    // Create Terminal Input
    sprintf(cmd,"iw dev %s scan -u\n",iface);

    // popen returns NULL when pipe or fork fail so we iterate over
that condition
    // and read the terminal output to file fp
    while(!(fp = popen(cmd,"r"))){
        exit(1);
    }

    // Line by line it reads the terminal output from the file to the
buffer
    while(fgets(buf,sizeof(buf),fp)){
        line_cnt++;
        // get MAC address
        if((strncmp(buf,"BSS",3)==0)){
            memcpy(mac_aux,buf+4,17);
            bss_cnt++;
            line_cnt = 0;
        }
        // get channel
        if((line_cnt == 2) && (strncmp(buf,"\tfreq: ",6) == 0)){
            frqaux = strtol(buf+7,NULL,10);
        }
        if(strncmp(buf+18,"OUI fe:ff:00",12)==0){
            nb_counter++;
            memcpy(nbs_a[nb_counter].nb_mac,mac_aux,17);    //save mac
            nbs_a[nb_counter].freq = frqaux;        //save freq
            strcpy(nbs_a[nb_counter].data,buf+38); //save data
        }
    }
    pclose(fp);

    //The type variable present in the beacon might be used to
    //specify different kind of messages in the beaocn frame

    //copiar os valores obtidos para a estrutura
    for(i=0;i<=nb_counter;i++){
        sscanf(nbs_a[i].nb_mac, "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx",
&nb[i].nb_mac[0],&nb[i].nb_mac[1],&nb[i].nb_mac[2],&nb[i].nb_mac[3],&n
b[i].nb_mac[4],&nb[i].nb_mac[5]);
    nb[i].freq = nbs_a[i].freq;
    //Go through the data field and extract the specific parameters
    cnt = 0;
    pch = strtok(nbs_a[i].data,delim);
    while(pch != NULL){
        if(cnt == 0)
            type = (unsigned int) strtol(pch,NULL,16);
        else if (cnt == 1)
            nb[i].hops_to_gw = (unsigned int) strtol(pch,NULL,16);
        else if(cnt > 1)
            nb[i].tree_chan_lst[cnt - 2] = (unsigned
int)strtol(pch,NULL,16);

        cnt ++;
        pch = strtok(NULL,delim);
```

```
    }
    }
    if(debug_level >=2){
        fprintf(stderr,"\nTotal BSSs:%d",bss_cnt);
        fprintf(stderr,"\nBSS WIFIX DUAL RADIO: %d\n",nb_counter+1);
    }

    return nb_counter;
}

int choose_neighbor(int nb_cnt, neighbour *parent){
    int i=0, hop_min = 10000, cnt = 0;
    struct nb_info *nbaux; //candidate list
    int clstsize = 0;

    if(nb_cnt < 0){
        fprintf(stderr,"COULD NOT FIND A NEIGHBOUR: nb_cnt == 0");
        return -2;
    }else{
        // 1st Criteria: HOP COUNT

        // Walk through the list of neighbors and get the lowest hop
        for(i=0;i<=nb_cnt;i++){
            if(nb[i].hops_to_gw < hop_min){
                hop_min = nb[i].hops_to_gw;
                fprintf(stderr,"\nMinimum number of hops found: %d\n",
hop_min);
            }
        i=0;
        for(i=0;i<=nb_cnt;i++){
            // Iterates over the neighbor list and points to the
neighbors with the lowest number of hops
            if(nb[i].hops_to_gw == hop_min){    // If the number of
hops the the lowest
                printf("Alocating new candidate\n");
                cnt++;
                if(clstsize==0){
                    nbaux = (struct nb_info *)malloc(sizeof(struct
nb_info));
                    clstsize ++;
                }
                else
                    nbaux = (struct nb_info
*)realloc(nbaux,cnt*sizeof(struct nb_info));

                hop_min = nb[i].hops_to_gw;
                nbaux[cnt-1] = nb[i];
            }
        }
    }
    // 2nd Criteria ...
    /*
     * Set channel usage as the Second Criteria
     */

    // Choose Parent
    if(cnt>0){ //Check if list not empty
        //Chosing first parent from list
        memcpy(parent, &nbaux[0], sizeof(struct nb_info));
```

```
        printf("\nA new parent was
chosen\nMAC:%.2x:%.2x:%.2x:%.2x:%.2x:%.2x\nHops:%d\nFreq:%d\n",parent-
>nb_mac[0], parent->nb_mac[1],parent->nb_mac[2],
parent->nb_mac[3],parent->nb_mac[4],parent->nb_mac[5],parent-
>hops_to_gw,parent->freq);
        for(i=0; i<parent->hops_to_gw; i++){
            printf("Up_nb Channels:%d\n ",parent->tree_chan_lst[i]);
        }
        printf("\n");
    }
    else{
        printf("\nList of neighbors is not NULL but no neighbor was
chosen!");
        free(nbaux);
        return -1;
    }
    fflush(stdout);
    free(nbaux);
    return 0;
}

int connect_parent(neighbour *parent){
    char cmd[100];
    char buf[256];
    FILE *fp;
    int ret;

    sprintf(cmd,"iw dev %s connect \"%s\" %d
%.2x:%.2x:%.2x:%.2x:%.2x:%.2x",iface[sta_id].name, SSID, parent->freq,
parent->nb_mac[0], parent->nb_mac[1],parent->nb_mac[2],parent-
>nb_mac[3],parent->nb_mac[4],parent->nb_mac[5]);
    do{
            fp = popen(cmd,"r");
    }while(fp == NULL);

    if(!fp){
        fprintf(stderr,"\nIncorrect parameters or too many files.\n");
        return EXIT_FAILURE;
    }
    while((fgets(buf,sizeof(buf),fp)) && (!gotParent)){
        if(strncmp(buf,iface[ap_id].name,5)==0){

            printf("SUCCESS ON CONNECT\n");
            gotParent = TRUE;
            ret = 0;
        }
        else if((strncmp(buf,"command failed",14)==0) ||
(strncmp(buf,"/bin/sh: cmd: not found",20)==0)){
            printf("RETURNING CONNECT ON ERROR!\n");
            gotParent = FALSE;
            ret = -1;
        }
    }

    pclose(fp);

    return ret;
}

void addDownNeighbour(tap_list_info * tplst, unsigned char *mac, char
brdg[], real_if *iface, neighbour *nb)
```

```
{
    //if there's a tap already, update ttl
    /*
     * DEBUG de inputs
     */
    //show_tap_list_info(tplst);
    if( (tplst->tap_actual = getTapByMAC(tplst, mac)) != NULL){
        if(memcmp(tplst->tap_actual->mac, nb->nb_mac,
6*sizeof(unsigned char)) != 0)//if frame doesn't come from up
neighbour (avoid reciprocal neighbouring),
            tplst->tap_actual->ttl = 5;
    }else{
        if(memcmp(mac, nb->nb_mac, 6*sizeof(unsigned char)) != 0){//if
frame doesn't come from up neighbour (avoid reciprocal neighbouring)
            insert_tap_node(tplst, mac, iface, brdg, 0);
        }
    }
}

void checkOldDownNeighbour(tap_list_info * tplist, unsigned char *mac,
char brname[], real_if *iface)
{
    //removes tap (case exists) if not my down neighbour anymore
    if( (tplist->tap_actual = getTapByMAC(tplist, mac)) != NULL){
        if(tplist->tap_actual->up_nb_flag == 0)//just making sure that
up neighbour's tap is not being removed
            remove_tap_node(tplist, tplist->tap_actual, iface, brname,
0);
    }
}
```

Appendix E.5

This appendix contains the hapd.c file

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#include "general.h"

#include "common/wpa_ctrl.h"
#include "utils/os.h"

#ifndef CONFIG_CTRL_IFACE_DIR
#define CONFIG_CTRL_IFACE_DIR "/var/run/hostapd"
#endif /* CONFIG_CTRL_IFACE_DIR */

#define OUI "feff00"
#define BEACON_INFO_SIZE 256

static const char *ctrl_iface_dir = CONFIG_CTRL_IFACE_DIR;

static char *ctrl_ifname;
static struct wpa_ctrl *ctrl_conn;
int attached = 0;

static struct wpa_ctrl * open_connection(const char * ifname){
    char *cfile;
    int flen;

    if (ifname == NULL)
        return NULL;

    flen = strlen(ctrl_iface_dir) + strlen(ifname) + 2;
    cfile = malloc(flen);
    if (cfile == NULL)
        return NULL;
    snprintf(cfile, flen, "%s/%s", ctrl_iface_dir, ifname);

    ctrl_conn = wpa_ctrl_open(cfile);

    free(cfile);

    return ctrl_conn;
}

static void close_connection(void){
    if(ctrl_conn == NULL)
        return;
     /*
      * If the connection was used for unsolicited event
      * messages, it should be first detached by calling
      *  wpa_ctrl_detach().
      */
    if(attached == 1){
        wpa_ctrl_detach(ctrl_conn);
        attached = 0;
    }
```

```
    wpa_ctrl_close(ctrl_conn);
    ctrl_conn = NULL;
}

static void hostapd_msg_cb(char *msg, size_t len)
{
    printf("%s\n", msg);
}

static int _wpa_ctrl_command(struct wpa_ctrl *ctrl, char *cmd, int
print){
    char buf[4096]; // Response Maximum Length
    size_t len;
    int ret;

    if(ctrl_conn == NULL){
        printf("Not connected to hostapd - Command dropped\n");
        return -1;
    }

    len = sizeof(buf) - 1;
    ret = wpa_ctrl_request(ctrl,cmd,strlen(cmd),buf,&len,
            hostapd_msg_cb);

    if (ret == -2) {
            printf("'%s' command timed out.\n", cmd);
            return -2;
        } else if (ret < 0) {
            printf("'%s' command failed.\n", cmd);
            return -1;
    }
    if (print) {
        buf[len] = '\0';
        printf("%s", buf);
    }
    return 0;
}

static inline int wpa_ctrl_command(struct wpa_ctrl *ctrl, char *cmd){
    return _wpa_ctrl_command(ctrl, cmd, 1);
}

static int hostapd_cmd_ping(struct wpa_ctrl *ctrl){
    return wpa_ctrl_command(ctrl, "PING");
}

static int hostapd_cmd_set(struct wpa_ctrl *ctrl, char *var, char
*val)
{
    char cmd[256];
    int res;

    res = os_snprintf(cmd, sizeof(cmd), "SET %s %s", var, val);
    if (os_snprintf_error(sizeof(cmd), res)) {
        printf("Too long SET command.\n");
        return -1;
    }
    return wpa_ctrl_command(ctrl, cmd);
}
/*
static int hostapd_cmd_beacon_update(struct wpa_ctrl *ctrl){
```

```
    return wpa_ctrl_command(ctrl, "UPDATE_BEACON");
}*/

/*
 * Construct the information element to be included in beacon frames
 * <id><len><oui><type><mesh_info>
 * Calculate the info field lenght = 3[oui] + 1 [type] + N[payload]
 *                                 =   dd+len+4
 */

int set_beacon_ie221(char *ifname, int hops, int chnl_lst[],int
parent_chnl){

    int ret = 0, i = 0;
    char beacon_info[25];
    char *binfo;

    // Set hops info
    sprintf(beacon_info,"dd%02xfeff0001%02x",5+hops,hops);
    // Set tree channel usage info
    for(i=0;i<(hops-1);i++){
        printf("\nDEBUG:chnl_lst[%d]:%d\n",i,chnl_lst[i]);
        sprintf(beacon_info,"%s%02x",beacon_info,chnl_lst[i]);
    }
    //
    if(!gw)
        sprintf(beacon_info,"%s%02x",beacon_info,parent_chnl);

    printf("\nIE(221):%s\n",beacon_info);
    ctrl_ifname = ifname;
    ctrl_conn = open_connection(ctrl_ifname);

    if(ctrl_conn){
        printf("\nLigação com hostapd estabelecida");
        // Teste à ligação com a interface de controlo do hostapd
        printf("\n--- SENDING PING ---\n");
        printf("HOSTAPD: ");
        ret = hostapd_cmd_ping(ctrl_conn);
        // Fazer o SET aos vendor elements e receber "OK"
        printf("\n--- SENDING SET ---\n");
        printf("HOSTAPD: ");
        ret = hostapd_cmd_set(ctrl_conn, "vendor_elements",
beacon_info);
    }

    else
        printf("\nUnable to open connection with Hostapd");
    close_connection();

    printf("\n");
    fflush(stdout);
    return ret;
}
```

Appendix E6.

This appendix contains the ofreard.c file

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/ether.h>
#include <netinet/in.h>

#include "forward.h"
#include "general.h"

int fwd_packet(tap_list_info *list_info, unsigned char frame[], int
frame_size)
{
    int written = -1;
    tap_node *dest_tap;

    dest_tap = getTapByMAC(list_info, frame+6);

    if (dest_tap != NULL){
        //increments pointer "frame" 14 positions(dst+src+ethtype),
erasing header
        if((written = write(dest_tap->fd, frame+14, frame_size-14)) ==
-1){
            fprintf(stderr,"TAP SOCKET WRITE FAILED %s:
%s\n",dest_tap->name, strerror(errno));
            return -1;
        }
        else{
            //---debug---
            if(debug_level >= 2)
                printf("frame(%d) sent to %s\n",written,dest_tap-
>name);
            if(debug_level == 3)
                show_frame("* FRAME SENT ", frame+14, frame_size-
14);//for showing purposes, the frame sent to tap is incremented 14
positions erasing the header
            //---end of debug
        }
        return 0;
    }
    else{
        //---debug---
        if(debug_level == 2)
            printf("Corresponding tap NOT CREATED yet\n");
        if(debug_level == 3)
            // Show frame without the ethernet header.
            show_frame("* Corresponding tap NOT CREATED yet",
frame+14, frame_size-14);
        //---end of debug
        return -2;
    }
}
```

```
int process_send(int if_fd, unsigned char *if_mac, unsigned char
*tap_dest_mac,  unsigned char *old_frame_buf, int old_frame_bufsize)
{
    int written = -1;
    //---debug---
    if(debug_level >=1){
        printf("\n\t--DEBUG:--\n\tFD: %d"
                "\n\tIF_MAC: %.2x:%.2x:%.2x:%.2x:%.2x:%.2x"
                "\n\tDST_MAC: %.2x:%.2x:%.2x:%.2x:%.2x:%.2x\n",
                if_fd,
if_mac[0],if_mac[1],if_mac[2],if_mac[3],if_mac[4],if_mac[5],

tap_dest_mac[0],tap_dest_mac[1],tap_dest_mac[2],tap_dest_mac[3],tap_de
st_mac[4],tap_dest_mac[5]);
        //---end of debug
    }

    struct ethhdr *old_frame_header, *nw_frame_header;
    int nw_frame_bufsize = 0;
    unsigned char* nw_frame_buf;

    //creates pointer to old_frame - structure ethhdr type
    old_frame_header = (struct ethhdr*) (old_frame_buf);
    //defines size for new frame
    nw_frame_bufsize = sizeof(struct ethhdr)+(old_frame_bufsize);
    //pointer to beginning frame to be sent
    nw_frame_buf = (unsigned char *)malloc(nw_frame_bufsize);
    //defines pointer to new frame header
    nw_frame_header = (struct ethhdr*) nw_frame_buf;

    //copies old frame to new as payload, notice the header increment
    memcpy(nw_frame_buf+sizeof(struct ethhdr), old_frame_buf,
old_frame_bufsize);

    //new frame header - Generic Routing Encapsulation (GRE)
    nw_frame_header->h_proto = htons(0xFFFF);
    //source mac_addr - 6 is the size of the if_mac array
    memcpy(nw_frame_buf+6, if_mac, 6*sizeof(unsigned char));
    //destination mac
    memcpy(nw_frame_buf, tap_dest_mac, 6*sizeof(unsigned char));
    // minimum frame size is 60 bytes(6+6+2+46) -
    // if frame is smaller it's filled with useless bytes
    // writes new packet on real interface
    if((written = write(if_fd, nw_frame_buf, nw_frame_bufsize)) == -
1){
        fprintf(stderr,"SOCKET WRITE FAILED: %s\n",strerror(errno));
        free(nw_frame_buf);
        return -1;
    }
    else{
        //---debug---
        if(debug_level >= 2)
            printf("frame(%d) sent from IF \n",written);
        if(debug_level == 3)
            show_frame("FRAME SENT", nw_frame_buf, nw_frame_bufsize);
        //---
    }
    free(nw_frame_buf);

    return 0;
}
```

Appendix E.7

This appendix contains the channel.c file

```
#include <stdio.h>
#include <stdlib.h>

int choose_channel(int tree_chnl_lst[], int parent_chnl, int hops){

    // The main focus of such a procedure is to minimize interference
throughout the branch
    // At the end the node will also pick the least active channel
attemping not only to
    // minimize interference with close neighbors but also chose a
favorable channel

    int bst = 0, i = 0, j = 0, channel = 0;
    // Non-overllaping channel list (16)
                            //{0,1, 2, 3, 4, 5, 6, 7, 8, 9,10, 11, 12,
13, 14, 15}
    unsigned int chnl_lst[] =
{1,6,11,36,40,44,48,52,56,60,64,100,104,108,112,116};
    float *candidates = malloc(sizeof(unsigned int));

    printf("\n### CHOOSING CHANNEL FOR NEXT HOP ###\n");

    // If parent is on a 5GHz channel allocate 2.4GHz candidates
    if(parent_chnl > 14){
        printf("\nChoosing from 2.4GHz channel list\n");
        candidates = (float *) realloc(candidates, 3 * sizeof(unsigned
int));
        //Every candidate starts with an equal maximum weight of
beeing picked
        for(i=0; i<3; i++){
            candidates[i] = 1;
        }
        printf("\nMY HOPs:%d \n",hops);
        // Run the channel usage list in hops of 2 so that we get only
the channels of interest
        for(i = hops-2 ; i >= 0; i-=2 ){
            printf("UPSTREAM CHANNELS: %d\n",tree_chnl_lst[i]);
             //printf("%d ",tree_chnl_lst[i]);
            //Check the element position in the list
            for(j=0; j < 3; j++){
                //search element position in the channel list
                if(chnl_lst[j] == tree_chnl_lst[i]){
                    //reduce the weight by an amount
                    //printf("%f = %f * ((%d-
%d))/%d\n",candidates[j],candidates[j],hops,i,hops);
                    candidates[j] = candidates[j] * ((hops-i)/2)/hops;
                }
            }
        }
    }
```

```
    // If parent is on a 5GHz channel allocate 2.4GHz candidates -----
---------

    else if(parent_chnl < 14){
        printf("\nChoosing from 5GHz channel list\n");
        candidates = (float *) realloc(candidates, 13 *
sizeof(unsigned int));
        for(i=0; i < 13; i++)
            candidates[i] = 1;
        for(i = hops-2 ; i >= 0; i-=2 ){
            //Check the element position in the list
            for(j=0; j < 13; j++){
                if(chnl_lst[j+3] == tree_chnl_lst[i])
                    //reduce the weight by an amount
                    candidates[j] = candidates[j] * ((hops-i)/2)/hops;
            }
        }
    }

    // Choose the most weighted candidate
    if(parent_chnl < 14){
        printf("\nCandidates: ");
        for(j=0; j < 13; j++)
            printf("%f ",candidates[j]);
        printf("\n");
        for(i = 0; i < 3; i++){
            if(i==0)
                bst = 0;
            else{
                if(candidates[i] > candidates[bst])
                    bst = i;
            }
        }
        channel = chnl_lst[bst + 3];
    }

    if(parent_chnl > 14){
        for(j=0; j < 3; j++)
            printf("%f ",candidates[j]);
        printf("\n");
        for(i = 0; i < 3; i++){
            if(i==0)
                bst = 0;
            else{
                if(candidates[i] > candidates[bst])
                    bst = i;
            }
        }
        channel = chnl_lst[bst];
    }
    printf("\n Best Channel for next hop: %d\n",channel);
    free(candidates);
    return channel;
}
```

Appendix F.

Injection script developed with *pyLorcon2*

```
#!/usr/bin/env python

import sys
import random

import PyLorcon2

from impacket import dot11
from impacket.dot11 import Dot11
from impacket.dot11 import Dot11Types
from impacket.dot11 import Dot11ManagementFrame
from impacket.dot11 import Dot11ManagementBeacon

def getBeacon(src, ssid):
  "Return a 802.11 Beacon."

  # Frame Control
  frameCtrl = Dot11(FCS_at_end = True)
  frameCtrl.set_version(0)

frameCtrl.set_type_n_subtype(Dot11Types.DOT11_TYPE_MANAGEMENT_SUBTYPE_
BEACON)
  # Frame Control Flags
  frameCtrl.set_fromDS(0)
  frameCtrl.set_toDS(0)
  frameCtrl.set_moreFrag(0)
  frameCtrl.set_retry(0)
  frameCtrl.set_powerManagement(0)
  frameCtrl.set_moreData(0)
  frameCtrl.set_protectedFrame(0)
  frameCtrl.set_order(0)

  # Management Frame
  sequence = random.randint(0, 4096)
  broadcast = [0xff, 0xff, 0xff, 0xff, 0xff, 0xff]
  mngtFrame = Dot11ManagementFrame()
  mngtFrame.set_duration(0)
  mngtFrame.set_destination_address(broadcast)
  mngtFrame.set_source_address(src)
  mngtFrame.set_bssid(broadcast)
  mngtFrame.set_fragment_number(0)
  mngtFrame.set_sequence_number(sequence)

  # Beacon Frame Frame
  beaconFrame = Dot11ManagementBeacon()
  #beaconFrame.set_timestamp(0)
  beaconFrame.set_beacon_interval(100)
  beaconFrame.set_ssid(ssid)
  beaconFrame.set_supported_rates([0x82, 0x84, 0x8b, 0x96, 0x0c, 0x18,
0x30, 0x48])
```

```
#beaconFrame._set_element(dot11.DOT11_MANAGEMENT_ELEMENTS.EXT_SUPPORTE
D_RATES, "\x12\x24\x60\x6c")

  mngtFrame.contains(beaconFrame)
  frameCtrl.contains(mngtFrame)

  return frameCtrl.get_packet()

  #def exit(self):


if __name__ == "__main__":
  if len(sys.argv) != 3:
    print "Usage"
    print "  %s  <iface> <essid>" % sys.argv[0]
    sys.exit()

  iface = sys.argv[1]
  essid = sys.argv[2]

  context = PyLorcon2.Context(iface)
  context.open_injmon()
  moniface = context.get_capiface()

  src = [0x00, 0x00, 0x00, 0x11, 0x22, 0x33]
  beacon = getBeacon(src, essid)

  if essid == "":
    essid = "broadcast"

  print "Using interface %s" % iface
  print "Injecting Beacons for '%s'." % essid

  context.send_bytes(beacon)

  context.close()

  iface.exit()
```

Appendix G.

Linux Kernel log on the STA after performing the Beacon vendor IE modification tests.

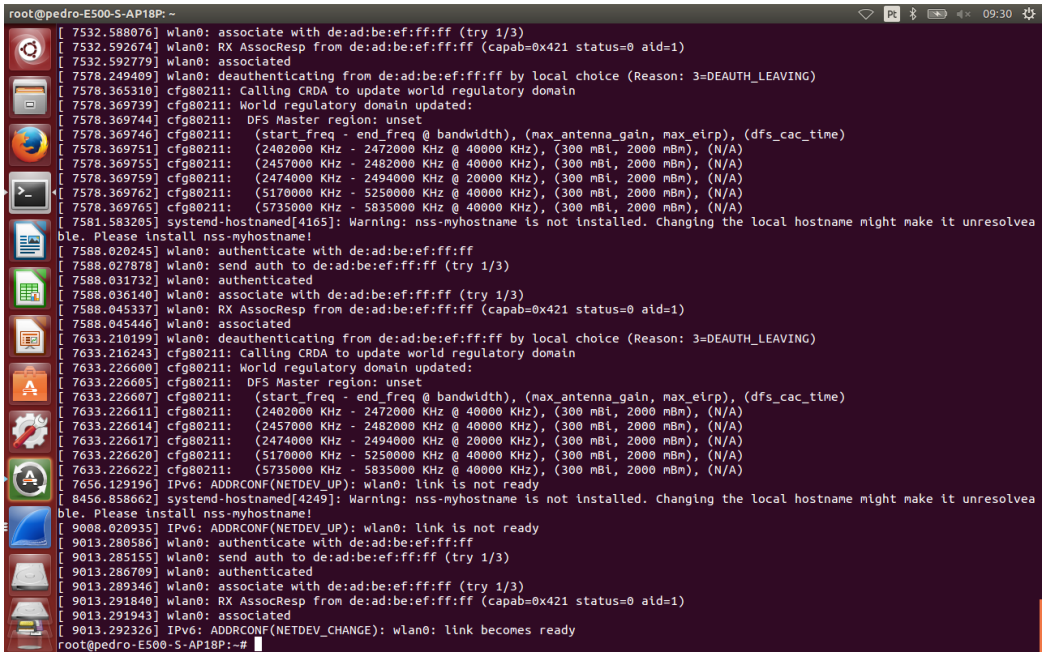


Figure 43 – STA kernel log

Appendix H.

The printscreen here shown concern the network topology as stated in Section 5.2.5

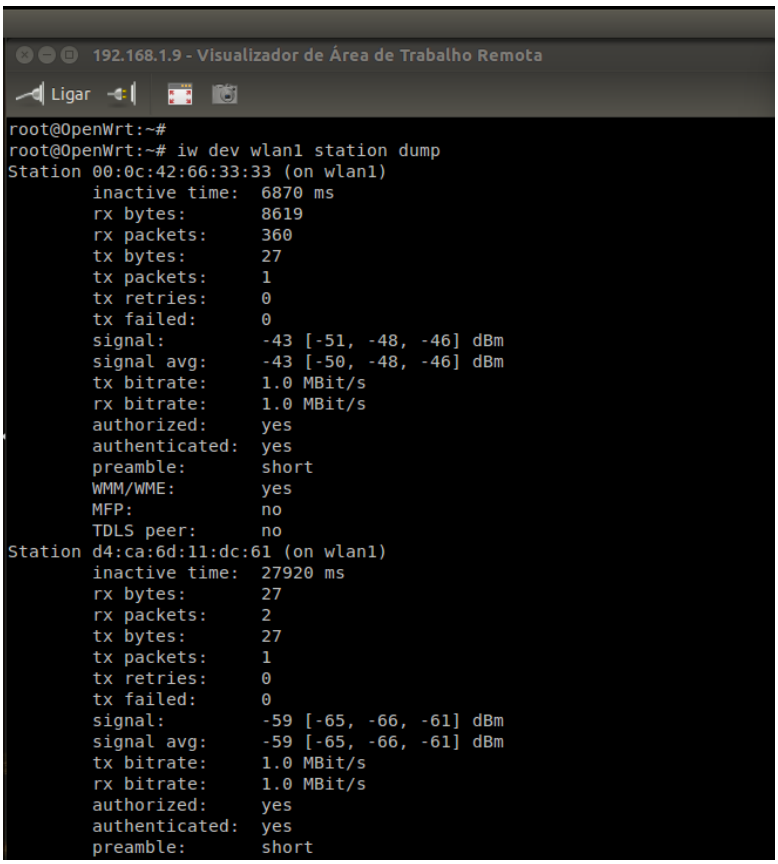


Figure 44 - GW with two stations at its parenthood

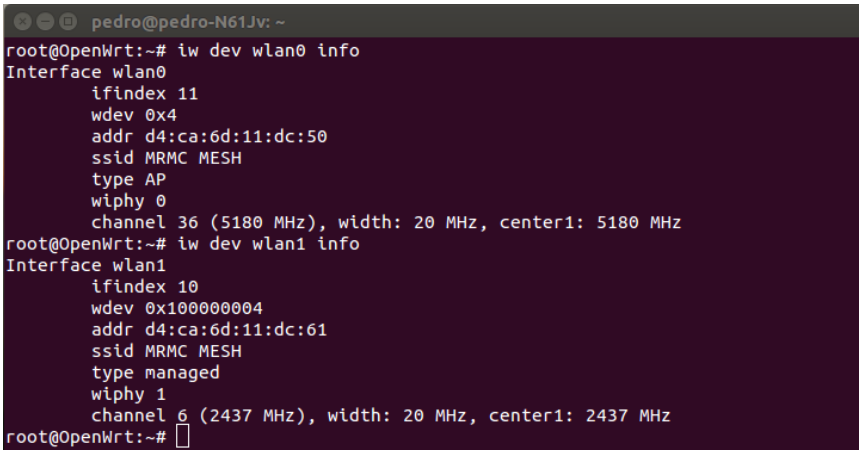


Figure 45 - MAP 1 connections. UP-NIC and DOWN-NIC

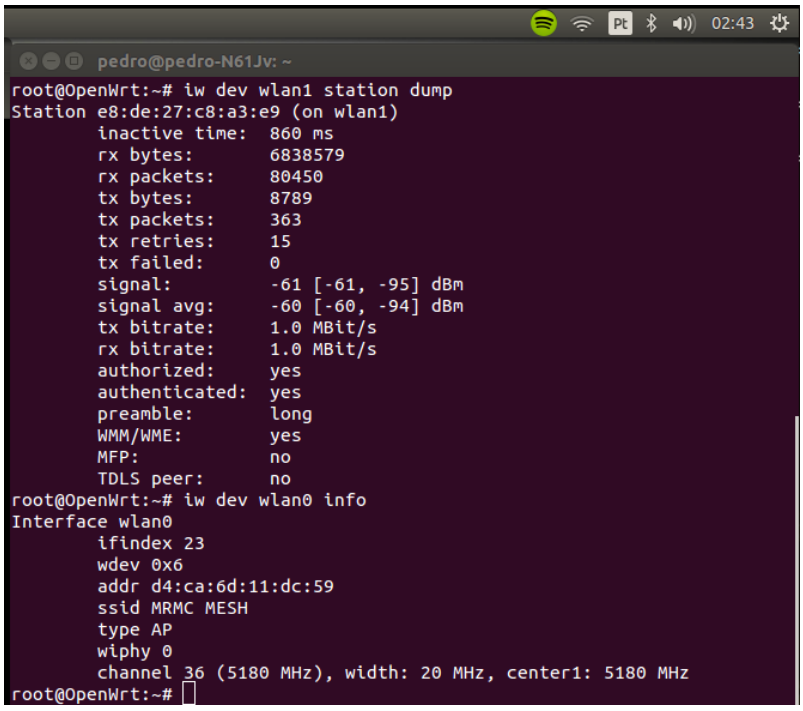


Figure 46 - MAP 2 Associated with the GW and with one station associated

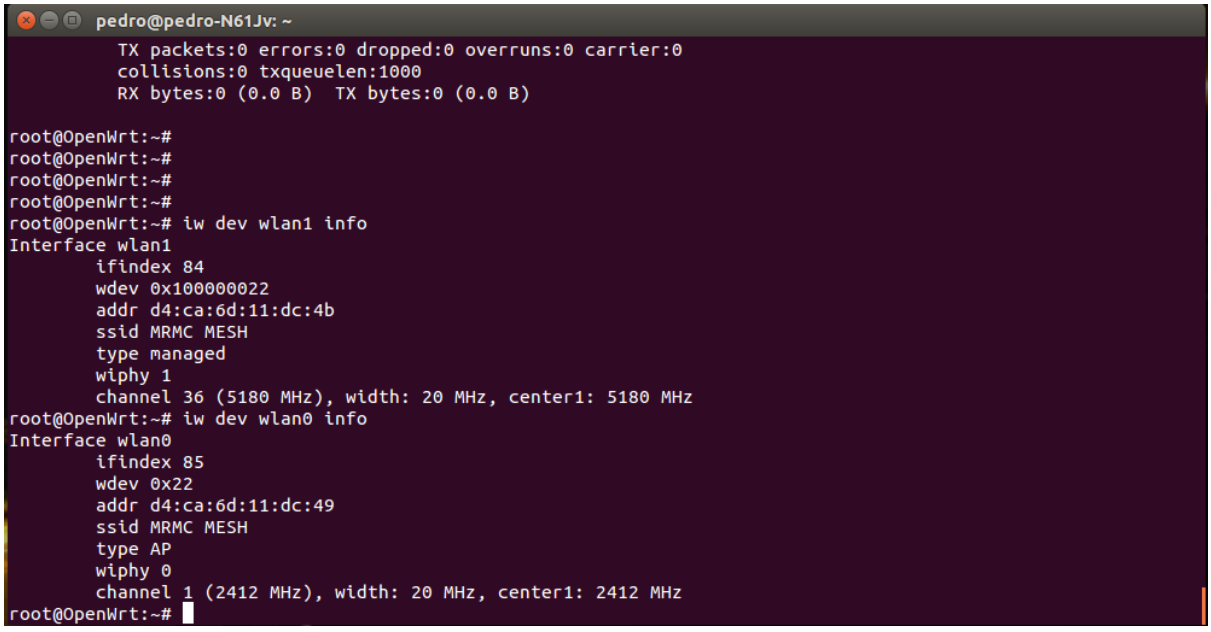


Figure 47 - MAP 3 with MAP 2 as its parent

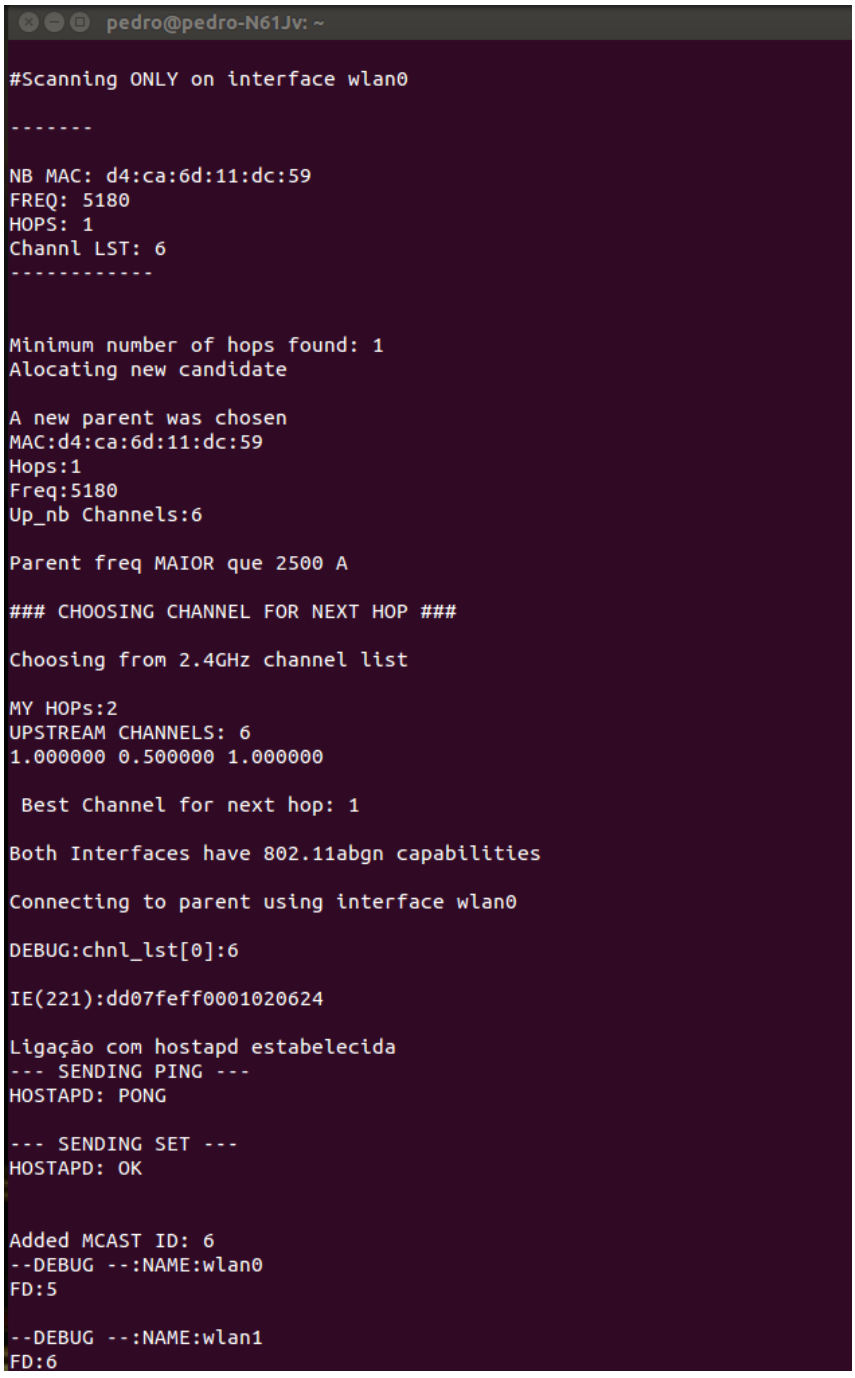


Figure 48 - MAP 3 log