Wireless Communications Project

Apostolopoulou Ioanna, Kofopoulos Lymperis Stamatis, Ntemkas Christos

June 2023

Contents

1 Abstract

In this project, we aim to implement the IEEE 802.11ac standard on a Wireless Mesh Network. We will compare two routing protocols: the reactive Modified Ad Hoc On-Demand Distance Vector Routing Protool and the proactive Optimized Link State Routing Protocol. This evaluation will take place within the Indoor Testbed of the NITlab. Due to limited hardware resources available on the Internet, there is a pressing need to fully utilize channel resources. Addressing this issue, we will explore channel allocation in multi-interface multi-channel Mesh nodes as a means to effectively improve network performance.

2 Mesh Network

A wireless mesh network is a network of devices (often called nodes) that communicate with each other using peer-to-peer wireless links typically over multiple hops – that is, by bouncing a message from one device, through another, and landing at a third (or fourth, etc.). It can also be a form of wireless Ad Hoc network. Wireless mesh networks often consist of mesh clients, mesh routers and gateways.

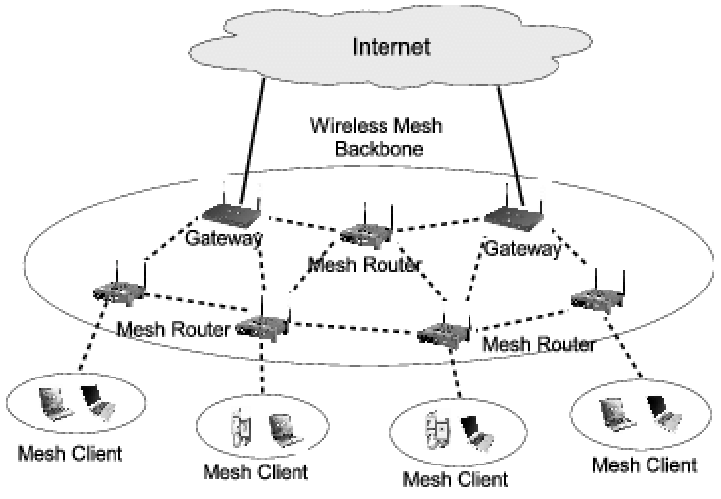


Figure 1: Wireless Mesh Network Example

The first challenge in mesh networking is the wireless link itself. Wireless communication technologies are locked in a multi-way trade-off between communication range, data rate (link throughput), cost, and size/power.

Another key challenge is how the wireless broadcast channel is shared between nodes — commonly referred to as medium access control (MAC). Unlike a centralized system such as a cellular network, a mesh network does not have a controller to hand out access rights. Nodes have to execute a decentralized algorithm to equitably share the channel. The obvious idea of each node sending at a random time results in too many collisions and poor performance.

Mesh networks are bandwidth-constrained and energy-constrained, especially those that trade-off data rate for range, and hence the growth of packet load as a function of size needs to be curtailed.

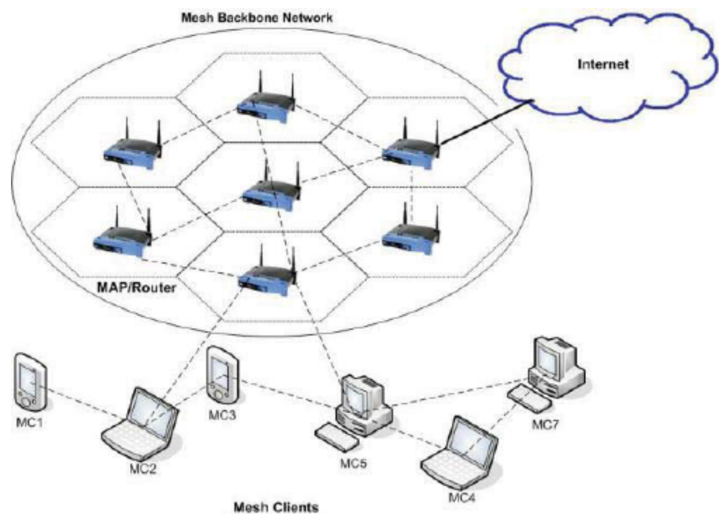


Figure 2: Wireless Mesh Network Example

3 Ad Hoc On-Demand Distance Vector Routing Protocol

3.1 AODV Routing Protocol

The Ad Hoc on-demand Distance Vector Routing Protocol, which is an improved model of DSDV Protocol, minimizes the number of Broadcast by creating Routes on-demand. The AODV protocol consists of two phases : i) route discovery and ii) maintenance phase.
The type of messages sent by AODV Routing protocol could be:

1. PREQ, Path Request
2. RREP, Route Reply
3. RRER, Route Error
4. RREP_ ACK, Route Reply Acknowledgement

So when one Node wants to communicate with one other it generates a PREQ packet and broadcasts it to its mesh neighbors. The main challenge is when the discovery phase is initiated. The Node that received the PREQ packet need to establish a temporary reverse route so it can reply to the previous node. Then it starts searching in its routing table if there is a valid route to the destination Node. If a valid routing path is found then the Node sends a RREP back to the source Node, otherwise continues to broadcast the RREQ packet. The destination Node sends a RREP packet when it receives the RREQ packet. Finally when the source receives the RREP packet it creates the routing path to the destination.

AODV depends on Route Discovery to calculate the shortest path to the destination, which in our implementation will be achieved by calculating the signal strength and creating a threshold of -42dBm as a lower bound of nodes considered close and in 1-hop distance.

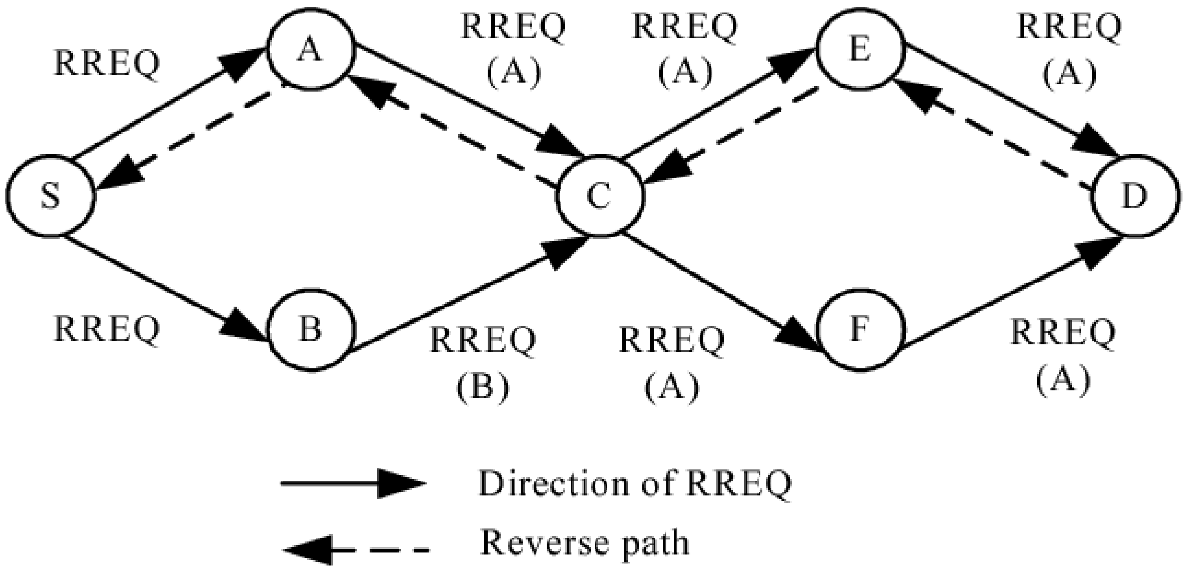


Figure 3: AODV Routing Protocol

3.2 Modified AODV Routing Protocol

The Modified AODV Routing Protocol fully utilizes channel and interfaces resources to establish efficient communication routes between nodes in a dynamic, self-configuring network. The protocol creates multiple interfaces for each node that communicate simultaneously. Alongside the multiple interfaces, the nodes can also use multiple communication channels. This is useful for reducing interference and increasing overall network capacity.

In Modified AODV, the process of discovering a route begins when a node within the network needs to transmit data to another node. This is accomplished by broadcasting a Path Request (PREQ) packet. The selection of the most suitable channel and interface for transmitting the PREQ packet is made by the node, considering factors such as the availability of channels, the existence of accessible interfaces and possible interference. Subsequently, the PREQ packet is disseminated throughout the network. Nodes that receive the PREQ packet analyze it to determine whether they possess a valid route to the destination. If they lack a route, they continue to propagate the PREQ packet.

Route establishment occurs when the PREQ packet reaches either the destination node or an intermediate node that possesses a valid route to the destination. In such cases, a Route Reply (RREP) packet is generated and sent back to the source node. Once the route is established, data packets can be transmitted from the source to the destination. The source node makes channel and interface selections for data transfer based on the established route, which in our implementation is achieved through the best metric of interfaces and channels.

Route Maintenance involves periodic monitoring of route quality. If a route becomes unreliable, the protocol initiates a route maintenance procedure to search for an alternative route.

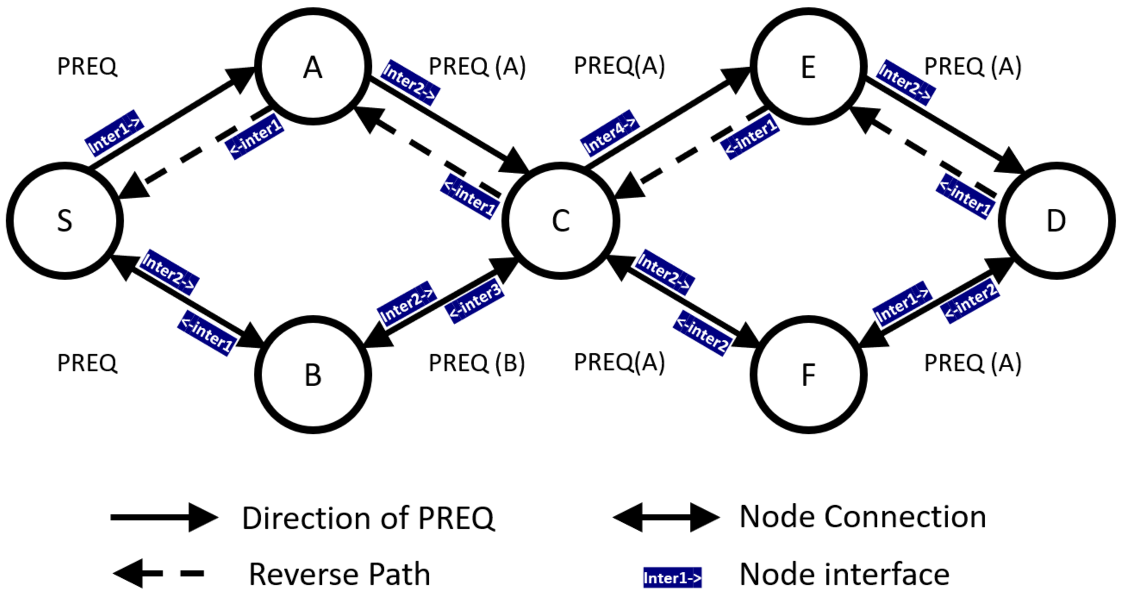


Figure 4: Modified AODV Routing Protocol

4 Optimized Link State Routing Protocol

OLSR is a proactive routing protocol, which means that each node in the network maintains a complete view of the network topology at all times. This is done by periodically flooding link state information throughout the network, and makes it very efficient at routing packets in mesh networks, which are often characterized by dynamic topologies.

However, OLSR optimizes the flooding process by using a technique called Multipoint Relays (MPRs). MPRs are selected nodes that are responsible for forwarding broadcast messages to other nodes in the network. This reduces the number of messages that need to be flooded and thus reduces the overall routing overhead.

Additionally, OLSR employs topology control technique to minimize the quantity of MPRs selected by each node. This is accomplished by employing a metric known as expected transmission count (ETX) to select MPRs with the most reliable links.

To route packets, OLSR uses a shortest path algorithm reliant on the continuously updated link state information that it maintains. This ensures that packets are routed along the shortest path between the source and destination nodes.

1. Each node in the network periodically broadcasts hello messages to discover its neighbors.
2. Each node calculates its MPR set using the ETX metric.
3. Each node periodically floods link state information to its MPRs.
4. Each node uses the link state information that it has received to construct its routing table.
5. When a node has a packet to forward, it uses its routing table to determine the next hop node.

This process is repeated continuously, which allows OLSR to maintain a complete view of the network topology and to route packets efficiently.

Overall, OLSR stands as a commendable choice for implementing routing in a mesh network, due to its efficiency, scalability and straightforward implementation.

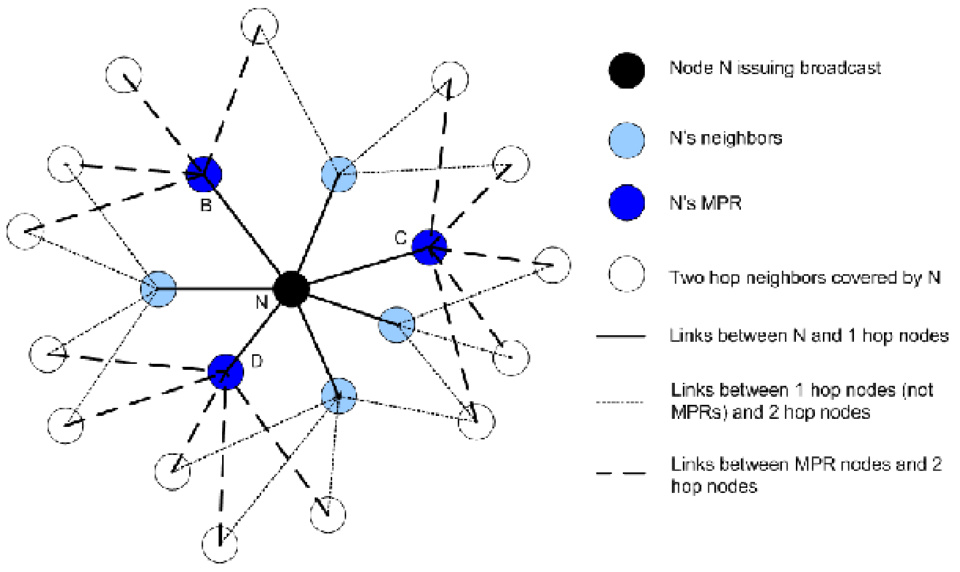

B
C
N
D
Node N issuing broadcast
N's neighbors
N's MPR
Two hop neighbors covered by N
Links between N and 1 hop nodes
Links between 1 hop nodes (not MPRs) and 2 hop nodes
Links between MPR nodes and 2 hop nodes

Figure 5: OLSR Routing Protocol

5 Experiment Setup

In this section we describe the steps on how to set the network in order to test the two routing protocols in the Indoor Testbed of NITlab.

5.1 Connecting to Testbed

1. Selection of Nodes form Nitos by using the Slice-Name
2. Connect to Nitlab3 Server with ssh:

   ```
   ssh SliceName@nitlab3.inf.uth.gr
   ```

3. Loading an OMF-compatible Image on my Resources:

   ```
   omf load -i baseline_wireless_communications.ndz -t node0X,node0X
   ```

   X indicates the number of node in the NITlab Testbed.

4. Turning ON the Nodes we will use :

   ```
   omf tell -a on -t node0X,node0X
   ```

5. In order to connect to the nodes we use ssh:

   ```
   ssh root@node0X
   ```

5.2 Initializing Driver

After we have connected to the Tesbed we need to Initialize the Drivers in each node we use. In order to accomplish that we have written the following script named init.sh. To run the following script we should enter the folder of the backports driver.

Specifically in the script we use the following commands:

1. Unload ath9k & ath10k drivers loaded.

```
modprobe -r ath9k

modprobe -r ath10k_pci
```

2. Create a configuration file for compile:

```
make defconfig-ath10k
```

3. Building the driver:

```
make && make install
```

4. Load the Driver in Kernel Memory

```
modprobe ath10_pci
```

5.3 Setting Up Mesh Network

Since the Driver is now loaded in the Kernel memory, it is know time to set the Mesh Network in the Testbed. The script we can use in each node is named mesh.sh and take as argument the wanted IP.

More specifically we use the folloing commands in the scripts:

1. Create an new interface for the Mesh Network & Join the Mesh Network wireless:

```
iw wlan0 interface add mesh0 type mp mesh_id wireless
```

2. Set Down the Wlan interface:

```
ifconfig wlan0 down
```

3. Set the Mesh interface Up & Set IP Address:

```
ifconfig mesh0 192.168.2.X up
```

   X indicates the number of node we are running the script.

5.4 Topology Used & Tx Power Setup

To establish the experimental environment, we utilize particular nodes that are specified in Figure 6, sourced from NITlab's Indoor Testbed, featuring the following characteristics:

1. Every node of our Mesh Network follows the IEEE 802.11ac standard.
2. Network Controller: Atheros 802.11ac 3x3.
3. Node 53 & Node 65 use Attenuators of 30 dBms and 20 dBms respectively.

In order to use the antena with the attenuator we run the folowing command:

```
iw phy phy0 set antenna 0x01 0x01
```

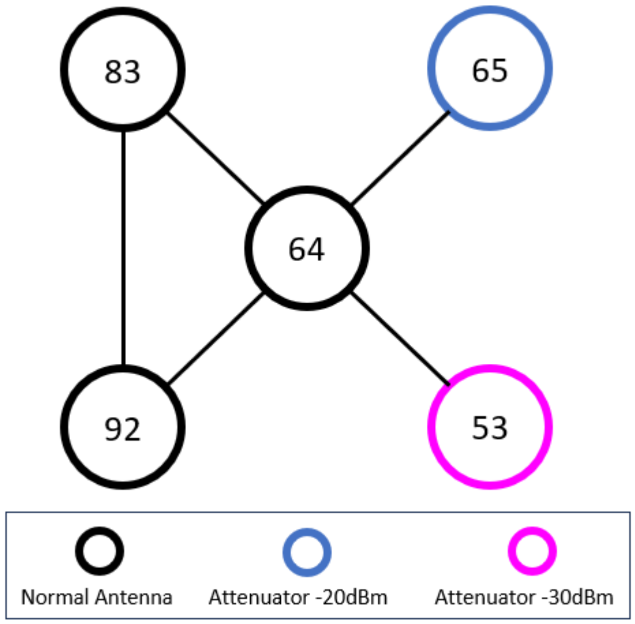


Figure 6: Our Topology

6 Implementation Features

6.1 Modified AODV Routing Protocol

In this project we implement multi interface multi channel AODV we had to make specific changes in the MAC Layer of the Atheros 10k Driver, using the IEEE 802.11ac standard.

6.1.1 Wireless Mesh Network & AODV Routing Protocol

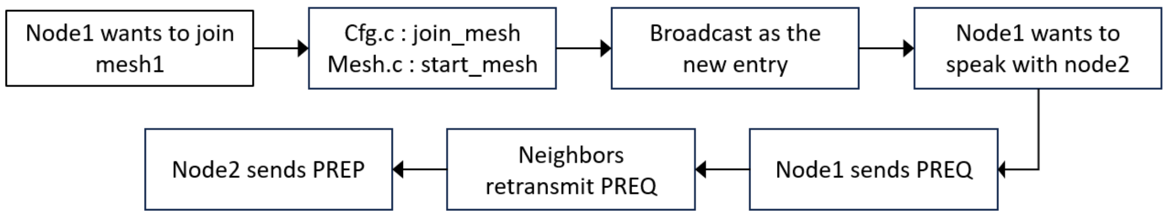

Node1 wants to join mesh1
Cfg.c : join_mesh Mesh.c : start_mesh
Broadcast as the new entry
Node1 wants to speak with node2
Node1 sends PREQ
Neighbors retransmit PREQ
Node2 sends PREP

Figure 7: Mesh Communication

In order to create the Mesh network every node of our topology needs to join the Mesh network. This procedure is starting in main.c where the operation .join_mesh in net/mac80211/cfg.c is defined as a member of "struct cfg80211_ops mac80211_config_ops". After the node initializes its presence in the mesh network using in /net/mac80211/mesh.c the function ieee80211_start_mesh(). Then the ieee80211_iface_work() function which can be found in /net/mac80211/iface.c initiates all processes regarding mesh frames. Moreover, function ieee80211_start_mesh() builds the beacon for mesh interface and starts broadcasting the node's existence in mesh network.

Each time a node needs to communicate with another node in the mesh, a PREQ is broadcasted to all of its neighbors. If the neighbor who received the PREQ is not the one that the initial node requested to communicate with the PREQ is retransmitted the same way to the next hop neighbors. On the other hand node that PREQ is searching is found the broadcasting is terminated in the destination node and the PREP is created from function mesh_path_sel_frame_tx() in /net/mac80211/mesh_hwmp.c.

All the type of messages send by AODV protocol are handled in /net/mac80211/mesh_hwmp.c. More specifically:

- In hwmp_preq_frame_process() the Path Request message is processed.
- In hwmp_prep_frame_process() the Path Reply message is processed.
- In case of Error the message is processed by the hwmp_perr_frame_process() function.

6.1.2 Modified AODV Routing Protocol for MIMC

In order to implement MIMC AODV we had to make changes in the /net/mac80211/mesh_hwmp.c in function hwmp_route_get_info() that update routing info to the originator and transmitter. This function takes as arguments the following:

- sdata: local mesh subif
- mgmt: mesh management frame
- hwmp_ie: hwmp information element (PREP or PREQ)
- action: type of hwmp ie

The function provides a metric to the originator of the frame or returns 0 if the PREQ frame has reached its destination. It begins by retrieving information about the station associated with the source address in the management frame and uses the last_hop_metric, which quantifies the link quality between the local mesh node and the source of the frame.

Additionally, it iterates through the list of interfaces of the node, associated with the local mesh network and calculates a temporary metric for the mesh path associated with the interface in current iteration. It compares the temporary metric to the lowest_metric found so far. If the current interface's metric is lower, it updates lowest_metric and records the best_sdata (interface with the lowest metric). This is done to determine the best interface for sending the PREP. After processing all interfaces, it unlocks the RCU read lock. The code assigns the best_sdata (interface with the lowest metric) as the chosen interface for sending the PREP. It re-looks up the mesh path for the original address with this chosen interface.

It then examines whether the mesh node is the original sender of the frame. If it is, the 'process' flag is set to false, and it's indicated that the information is no longer current. If the source is not the local mesh node, the function checks if there's an existing mesh path for the source address. In such a case, it updates the path based on various conditions, including whether the path is fixed, active, and whether the sequence number is newer, following the principles of the MIMC AODV protocol as shown in the Figure below.

If there's no pre-existing path for the source address, a new mesh path is established. If the information is still considered current at this stage, the function proceeds to update the mesh path with the newly calculated metric and other relevant details. The function also maintains and updates information pertaining to the routing of the data to be transmitted.

We also altered the /net/mac80211/mesh_plink.c function __mesh_sta_info_alloc() that returns the new allocated sta info and takes as arguments the flowing:

- sdata: local mesh subif
- hw_addr: mac of the station to be allocated
- rx_status: a struct containing the signal of the packet received

It's important to mention that the rx_status is included as an argument to assess the signal strength of the received packet. This evaluation helps decide whether to allocate this station, effectively creating a new station, or not. The threshold selected -42 dBs, was determined after experiments conducted.

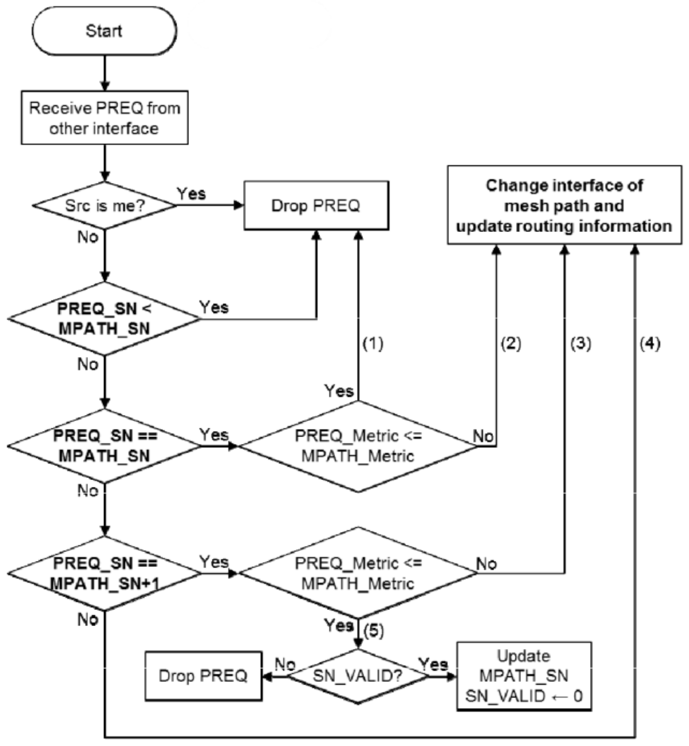

Start
Receive PREQ from other interface
Src is me?
Yes
No
Drop PREQ
Change interface of mesh path and update routing information
PREQ_SN < MPATH_SN
Yes
No
(1)
(2)
(3)
(4)
Yes
PREQ_SN == MPATH_SN
Yes
PREQ_Metric <= MPATH_Metric
No
No
PREQ_SN == MPATH_SN+1
Yes
PREQ_Metric <= MPATH_Metric
No
No
Yes
(5)
Drop PREQ
No
SN_VALID?
Yes
Update MPATH_SN SN_VALID ← 0

Figure 8: Block ACK Establishment

In the NITlab Testbed we observed that between nodes, whose distance was the minimum, the signal power measured was around -27 dBs (for example node063, node064), whereas nodes that the distance between them was reasonable, the signal power reached the -39 dBs (for example node063, node092). In our implementation of the infrastructure this was crucial for blocking the communication between nodes with attenuators and those far from them.

In case the sta is allocated, we create a new virtual interfaces so the communication is achieved only through this one interface. The new interface needs a name, a mac and a frequency of 5GH band. In the code we check how many neighbors exist until now and respectively we change the previous parameters.

6.1.3 Establishing Block ACK Session in Multi-hop

The initiation of a session is accomplished by exchanging an ADDBA (Add Block ACK) frame, which was already implemented in the driver. In the process of achieving this we firstly chech if there is another AMPDU in the /net/mac80211/status.c file by using the ieee80211_send_bar(). This function takes as arguments the:

- vif: struct ieee80211_vif pointer from the add_interface to create virtual interfaces
- addr: the peer's destination address
- tid: the traffic identification of aggregated session
- ssn: the new starting sequence number for the receiver

The functionality of the previous function is checked because in the function sta_info_alloc(), which can be found in net/mac80211/mesh_plink.c we initialize init_work in the net/mac80211/sta_info.c for the function ieee80211_ba_session_work(). In function ieee80211_tx_ba_session_handle_start() which is called in net/mac80211/ht.c and is implemented in /net/mac80211/agg-tx.c we start transmission

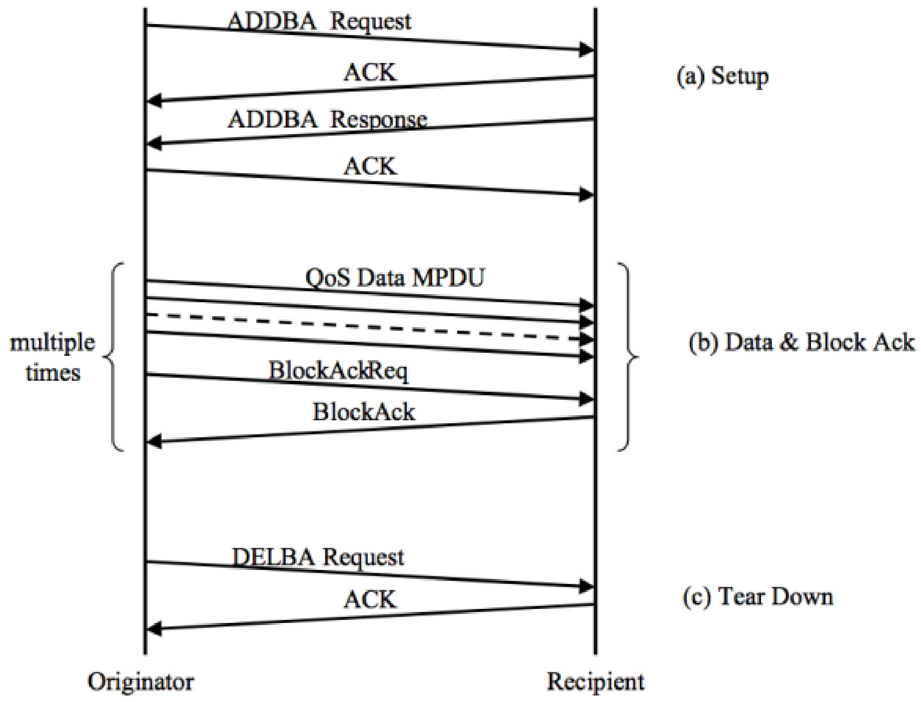


Figure 9: Block ACK Establishment

since we enter the if because not always *blocked* $==$ *true* or *tid_tx* $==$ $NULL$.

The DELBA Request is sent if one of the following happens:

- ieee80211_stop_tx_ba
- ___ieee80211_stop_rx_ba_session
- ieee80211_rx_reorder_ampdu

6.1.4 Management Sequence Number of MPDU

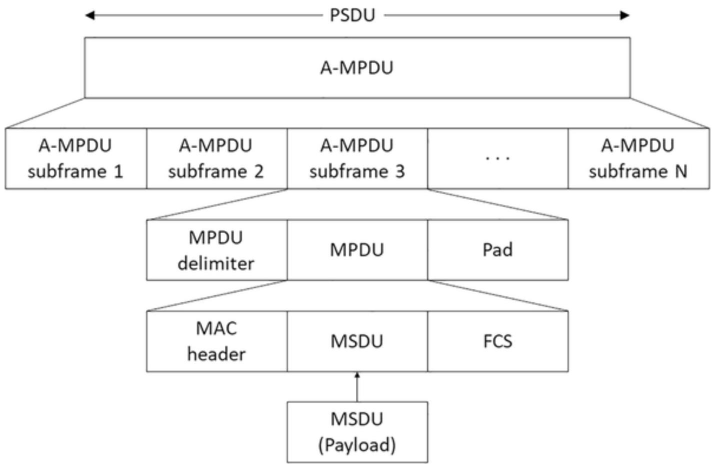


Figure 10: A-MPDU Frame

In order to create the AMPDU as it is shown in Figure 10, in the path net/mac8021/tx.c the following procedure is implemented.

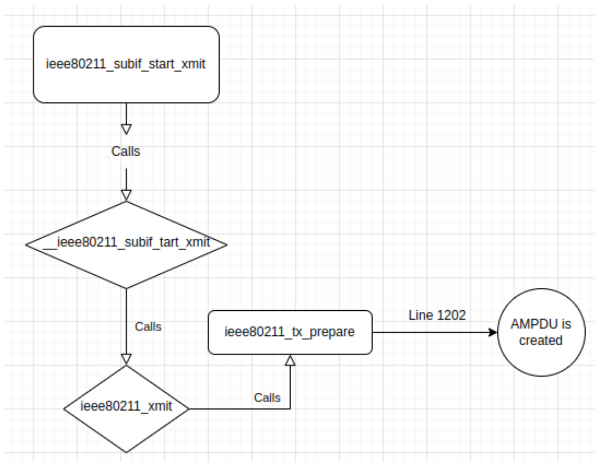


Figure 11: A-MPDU Creation procedure

In the majority of situations, MPDUs are sequentially aggregated when forming an A-MPDU. As a result, a minimal reordering algorithm becomes necessary for the recipient when handling re-transmitted MPDUs. This is implemented in net/mac80211/rx.c as shown in the Figure below.

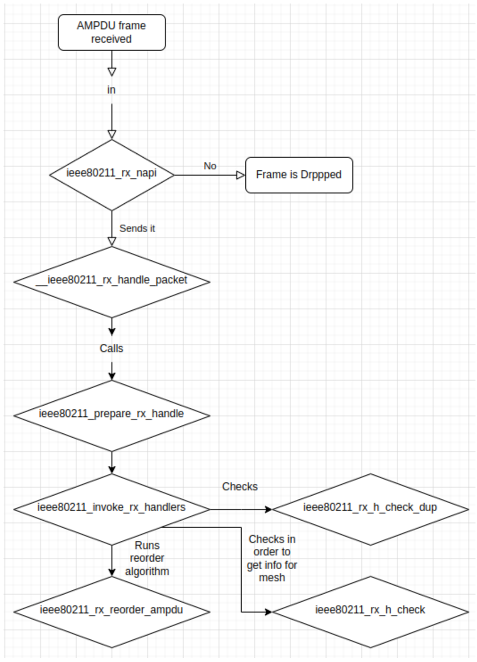

AMPDU frame received
in
ieee80211_rx_napi
No
Frame is Drppped
Sends it
__ieee80211_rx_handle_packet
Calls
ieee80211_prepare_rx_handle
ieee80211_invoke_rx_handlers
Checks
ieee80211_rx_h_check_dup
Runs reorder algorithm
Checks in order to get info for mesh
ieee80211_rx_reorder_ampdu
ieee80211_rx_h_check

Figure 12: A-MPDU Chech Dup & Reorder

This way is secured from Atheros 10k driver that duplicates of AMPDUs Frames are removed and in the final packet received the AMPDUs are in the right order.

6.2 OLSR Routing Protocol

In this part of the project we implemented Optimized Link State Routing Protocol for Ad Hoc Networks, by making specific changes in the MAC Layer of the Atheros 10k Driver, using the IEEE 802.11ac standard. It's important to note that in this implementation we assume that all links inside the mesh network are bidirectional.

6.2.1 Switching from AODV to OLSR algorithm

To stop AODV protocol and be able to switch to OLSR protocol, in function ieee80211_mesh_rx_mgmt_action() of the file mesh.c the switch case "case WLAN_CATEGORY_MESH _ACTION" had to be removed which is responsible for the functions processing the preq and prep and most importantly the hwmp_route_info_get(), where all the routing operations are taking place. By removing this case, the following process will stop: ieee80211_mesh_rx_mgmt_action() calls mesh_rx_path _sel_frame() which calls hwmp_route_info_get() as well as all the functions for processing :

- PREQ Path REQuest
- PREP Path REPly
- PERR Path ERRor
- RANN Route ANNouncement

Also we removed the call for the functions mesh_path_start_discovery() and ieee80211_mesh_rootpath() in mesh.c file inside the function ieee80211_mesh_work().

As for transmitting packets the ieee80211_build_hdr() is changed inside tx.c. This function is responsible for completing ieee80211 header. There if the CPTCFG_MAC80211_MESH was defined (in our case it was) the packet has mesh header. In case of AODV this header is made using the mpath which is updated in hwmp_route_info_get(), but for the case of OLSR the mesh header is filled based on OLSR routing table using OLSR_route_info_get() function.

6.2.2 Our Data Structures of OLSR

Struct neighbor holds the address of the neighbor node.

```
struct neighbor
{
    u8 addr[ETH_ALEN];
};
```

Struct Neighbor Table creates an array of neighbors based on their distance from the selected node. Additionally, it keeps track of the number of neighbors in 1 and 2 hop count and the MPR nodes we will use later.

```
struct neighbor_table
{
    struct neighbor *neighbors_1_hop;
    struct neighbor *neighbors_2_hop;
    struct neighbor *neighbors_MPR;
    int num_neighbors_1_hop;
    int num_neighbors_2_hop;
    int num_neighbors_MPR;
};
```

Struct Topology Table Entry is created to keep track of potential destination nodes, the previous node from path and the MPR sequence number. When we want to expire topology control messages we use the holding_time value (or htime elsewhere) of this struct, which counts the time through hops.

```
struct topology_table_entry
{
    u8 potential_dest[ETH_ALEN];
    u8 last_hop[ETH_ALEN];
    u32 MPR_seq_num;
    u32 holding_time;
};
```

Struct Routing Table Entry creates the Routing Table of the Driver by keeping the hop count, the next hop address and the source address.

```
struct routing_table_entry
{
    u8 addr[ETH_ALEN];
    u8 next_hop[ETH_ALEN];
    u32 hop_count;
};
```

Both structs of topology table and routing table contain the information of their entries and the number of their entries.

```
struct topology_table
{
    struct topology_table_entry *entries;
    int num_entries;
};

struct routing_table
{
    struct routing_table_entry *entries;
    int num_entries;
};
```

The OLSR Frame Type structure is used to check the type of OLSR message we received.

```
enum olsr_frame_type
{
    OLSR_HELLO = 0,
    OLSR_TC
};

struct OLSR_HELLO_message
{
    enum olsr_frame_type type;
    u32 htime;
    u32 willingness;
    struct neighbor *neighbors;
};

struct OLSR_TC_message
{
    enum olsr_frame_type type;
    u8 src_addr[ETH_ALEN];
    bool is_request;
    struct neighbor *neighbors_to_forward;
    int hops;
    u32 htime;
};
```

6.2.3 Neighbor Sensing

Each node in the network needs to identify its neighboring nodes with whom it has a direct two-way connection. To achieve this, nodes periodically broadcast HELLO messages that contain information about their neighbors and the status of those links. These control messages are sent in broadcast mode, received by all immediate neighbors, but not relayed further. In order to implement this we

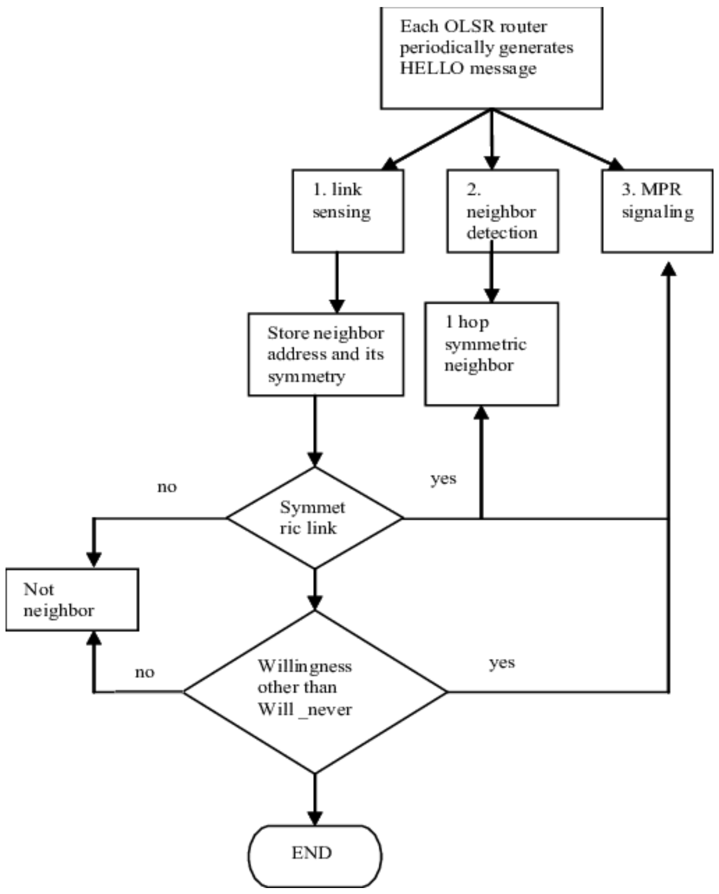

Each OLSR router periodically generates HELLO message
1. link sensing
2. neighbor detection
3. MPR signaling
Store neighbor address and its symmetry
1 hop symmetric neighbor
no
yes
Symmet ric link
Not neighbor
no
Willingness other than Will _never
yes
END

Figure 13: OLSR HELLO Message Procedure

have created the following functions in Atheros 10k Driver using the IEEE 802.11ac standard. More specifically:

1. OLSR_send_HELLO (struct ieee80211_sub_if_data *sdata);
   Is responsible for creating and sending the HELLO message, using as argument the sdata, which shows us the local mesh subif. This function is called and starts in ieee80211_mesh_housekeeping() function in mesh.c. We chose this function to call the OLSR_send_HELLO() due to its periodicity.

2. olsr_hello_process(struct ieee80211_sub_if_data *sdata, struct ieee80211_mgmt *mgmt, struct sk_buff *skb);
   This function processes the the skb of the HELLO message and extracts information regarding the neighbors. Checks if neighbors are already in the neighbor table, otherwise adds them to it.

3. add_neighbor(u8 addr[ETH_ALEN], char* type);
   In order to add the neighbors in the neighbor_table the olsr_hello_process() calls this function. Specifically the add_neighbor function adds a new neighbor to the table based on the type of it.

6.2.4 Topology Control

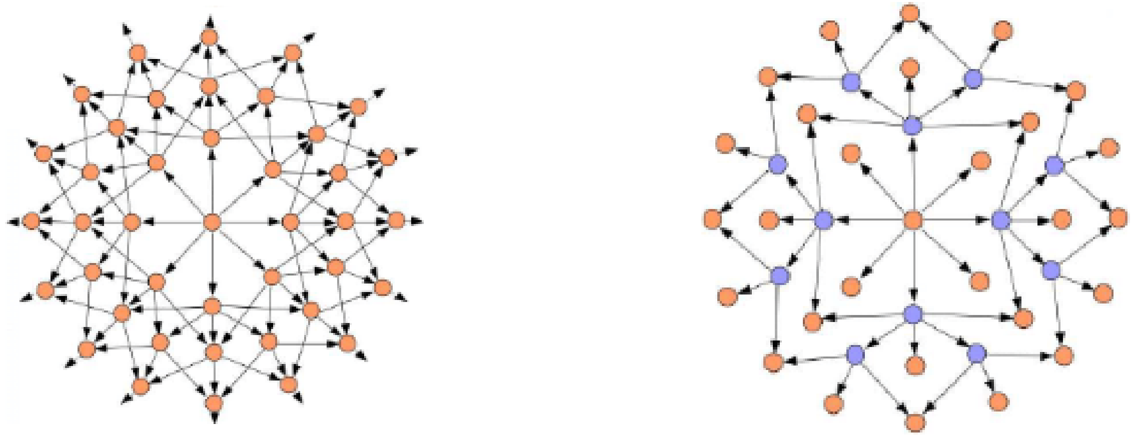
In OLSR, topology control is accomplished by exchanging TC (topology control) messages. The implemented functions for this operation:

1. OLSR_send_TC() (tx.c) This piece of code send and forwards the TC messages across the network. The TC message contains the MPR neighbors if there are any, or the 1 hop neighbors.

2. olsr_tc_process() (mesh_hwmp.c) This function is responsible for extracting the data of a TC message. If the neighbor sending this is not MPR, we drop the frame. Additionally, if the holding time is about to expire, we add a entry in both routing and topology table.

6.2.5 Multipoint Relay Selection

MPRs are the neighbors responsible for forwarding packets. They are being selected as the least 1 hop neighbors that can forward a packet to any of the 2 hop neighbors. In order to accomplish that, the function calculate_MPR_neighbors() in mesh_hwmp.c, is used. In that function both the calculation and the recalculation of the MPR table take place. To calculate the MPR table, for each 1-hop neighbor a search is done for all the 2-hop neighbors it is linked to. If there is at least one new 2-hop neighbor, the 1-hop neighbor is added to the MPR table and every 2 hop neighbors it is linked to is been excluded. This way, in each iteration the 2 hop neighbors are eliminated and their communication is satisfied by the least 1 hop neighbors. This function is called during the calculation the routing table.



(a) Flooding Messages by all Nodes (b) Flooding Messages by MPR Nodes

Figure 14: Flooding OLSR Messages

6.2.6 Routing Table Calculation

Each node maintains a routing table to enable the forwarding and the transmission of packets to other destinations in the mesh network. The above is implemented by creating a new function in the MAC Layer of the driver.

The routing table is recalculated when a change in the network neighborhood is detected. More importantly, this recalculation does not generate or trigger the transmission of packets throughout the network or its one-hop neighborhood.

In this implementation we created the recalculate_routing_table() function in mesh_hwmp.c which firstly re-calculated the MPR table of the current node and empties the routing table so there are no outdated entries in it. The insertions start from 1 hop with h=1 and gradually increase the h value so all the routes are added sorted by the closest neighbor to the most far one. The add_route function fills up entry by entry the routing table.

References

[1] Won-Suk Kim & Sang-Hwa Chung (2012) Design and Implementation of the IEEE 802.11n in Multi-hop over Wireless Mesh Networks with Multi-Channel Multi-Interface.

[2] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, L. Viennot (2001) Optimized Link State Routing Protocol for Ad Hoc Networks.

[3] Sheng Liu, Yang Yang, Weixing Wang (2013) Research of AODV Routing Protocol for Ad Hoc Networks.

[4] Chen Lijuan (2010) Research on Routing Protocol Applied to Wireless Mesh Networks.

[5] Ian D. Chakeres & Elizabeth M. Belding-Royer (2004) AODV Routing Protocol Implementation Design.

[6] Yuhua Yuan & Hui-Min Chen & M. Jia (2005) An Optimized Ad-hoc On-demand Multipath Distance Vector(AOMDV) Routing Protocol

[7] Anbao Wang & B. Zhu (2014) Improving MPR Selection Algorithm in OLSR Protocol Based on Node Localization Technology.

[8] Yang Junmo & Kazuya Sakai & Bonam Kim & Hiromi Okada (2006) Cost-Aware Route Selection in Wireless Mesh Networks.

[9] Shams Qazi & Yi Mu & Willy Susilo (2009) Securing Wireless Mesh Networks with Ticket-Based Authentication.

[10] Zhong Hui & Pu ShengYuan (2017) Analysis and Research on OLSR Protocol for Multi-Channel Assignment of Wireless Mesh Network.

[11] Won Hyoung Lee & Ho Young Hwang(2019) A-MPDU aggregation with optimal number of MP-DUs for delay requirements in IEEE 802.11ac.