Mini Project - Group 1, 2020

Group 1, Aarhus University

Anton Sakarias Rørbæk Sihm	201504954
Peter Marcus Hoveling	201508876
Michele Craighero	202000774

IOT WiFi Mesh networking

Github: https://github.com/201508876PMH/IOT-Mesh-network

Guidance counselor: Nestor Javier Hernandez Marcano
Due date: 21-12-2020

Contents

1 Introduction **2**

2 Background **3**
2.1 Wireless Mesh Network . . . 3
2.2 Meshmerize . . . 4

3 Design **5**
3.1 What data is needed . . . 5
3.2 How is data fetched from a client . . . 5
3.3 How often should the data be fetched . . . 6

4 Implementation **6**
4.1 Node implementation . . . 6
4.2 Client implementation . . . 7

5 Test setup **8**
5.1 Prerequisite nodes . . . 8
5.2 Prerequisite client . . . 8

6 Experiments **8**
6.1 Dynamically plotting nodes . . . 9
6.2 Dynamically plotting shortest path . . . 9
6.3 Changing topology . . . 10

7 Conclusion **10**

8 Future work **11**

9 Bibliography **12**
Websites . . . 12

1 Introduction

A wireless mesh network is a decentralised type of wireless network. Often made up of multiple nodes and organised in a mesh topology, every node can interconnect to create a WLAN ad-hoc mesh network[5]. The network does not rely on pre-existing infrastructure, such as routers in wired networks or access points in managed wireless networks[4].

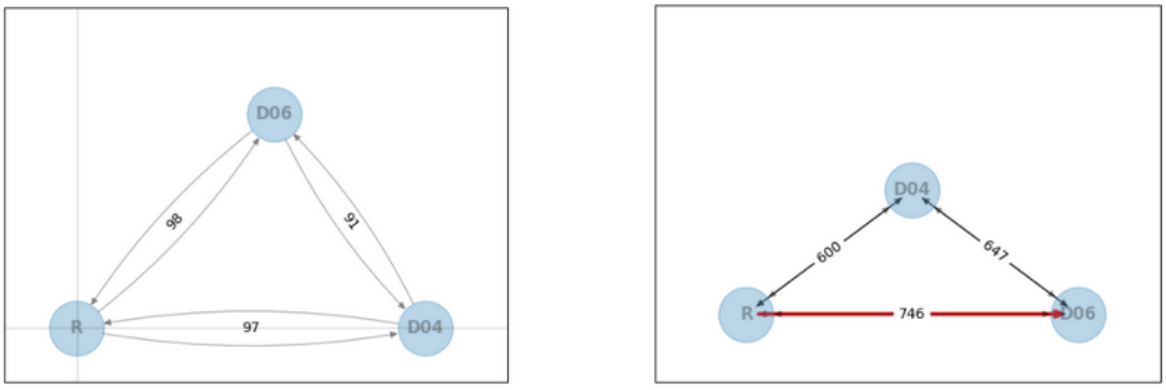
Mesh networking stems from the IEEE 802.11s wireless LAN standard and an IEEE.802 amendment[3]. The young Dresden based startup **Meshmerize**, is working on developing wireless mesh networks for industrial and critical applications and tries to further expand on the IEEE 802.11s protocol. They argue on their website, that for the last 40 years, wireless mesh networks have been consistently using single-path based routing protocols. Meshmerize tried to combat this by unleashing the devices from their imaginary cables through the novel multi-path routing approach[1].

Understanding what a wireless mesh network is and what meshmerize is, we take a look at project 5: WiFi mesh networking proposal. The project aims to give a deeper understanding, through research and exercise, in the following:

- Characterise WiFi mesh networking based on off-the-shelf network components from Meshmerize
- Define a set of performance metrics and conduct experiments to characterise the performance
- A comparison between different ad hoc routing schemes is included in the project preferable backed with quantitative investigation if the time allows

For the above listed goals, a meshmerize evaluation kit has been given. In this kit, 2 GL-iNet USB150 (black pen drives), 1 GL-iNet AR150 (router), 1 Portable USB Battery pack, 1 USB/A to micro-USB cable and 1 manual is given to set up this project.

Having completed the project, the results were the dynamic plotting of nodes on the basis of the nodes signal strength and the dynamic plotting of the meshmerize's chosen shortest path, as seen in figure 1. The rest of this report will be structured around how these results came to be.
More precisely, the rest of the paper is structured as follows: section 2 gives a general background on wireless mesh networks and Meshmerize protocol; section 3, 4 5 and 6 presents respectively design, implementation, test setup and the experiments done in our project; section 7 and 8 contain the conclusion and future works on the mesh protocol and at last section 9 has the bibliography.



(a) Dynamically plotted nodes

(b) Dynamically plotted shortest path

Figure 1: Images of some of the plotted graphs

2 Background

A mesh network is a network topology where the infrastructure nodes are connected in a dynamic and non-hierarchical way to many other nodes. A key aspect of a mesh network architecture is indeed the presence of multi-hop links and the possibility to use intermediate nodes to send packets to the others.
Mesh networks can be divided basing on the number of connections between nodes (partially or fully connected mesh networks) and also depending on the nature of the links in the network (wired or wireless mesh networks).
In this section we will focus on wireless mesh networks and then in particular on the Meshmerize protocol.

2.1 Wireless Mesh Network

Differently from a traditional network, where all devices are connected to a central access point, in a wireless mesh network the devices, or nodes, are directly connected to each other.

To better understand this aspect, we can refer to the example in Figure 2, in which there is a comparison between a traditional wireless network and a wireless mesh network.

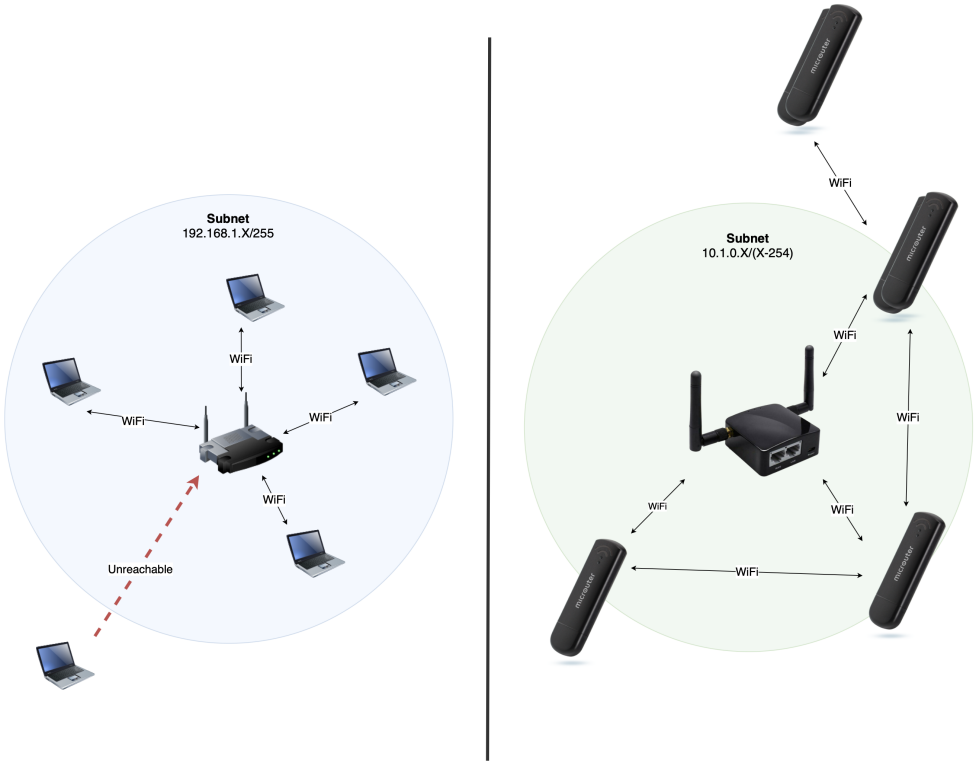


Figure 2: Traditional vs. Mesh Network

On the left, it is shown a traditional wireless network organised in a star-shape topology, where all devices are connected to the router and there are no direct connections between the other devices. In this case, a device can communicate with another one only using an indirect connection that passes through the central node (the router). This results in the fact that, if a device is out-of-range for the router (indicatively 100 m in an outdoor environment on the 2.4 GHz band), it becomes unreachable and can't communicate with the router and so neither with the other devices of the network.

On the right side of Figure 2, instead, it is shown an example of a wireless mesh network (partially connected): in this case the devices are directly connected with each others and this gives the possibility to communicate and send packets to other nodes of the network without passing through the router.
An important advantage of this type of networks is the fact that a device can be out-of-range for the router but still communicate with the other devices of the network and to the router itself: thanks to the presence of multi-hop links it can send packets to the destination passing through intermediate nodes. This characteristic makes wireless mesh network suitable also for wireless coverage in large indoor or outdoor environments and thus for many different applications.

Wireless mesh networks have two important features: the capacity to self-form, that significantly reduces the configuration costs of a traditional network, and to self-heal, that increases the resilience of the network. Since a node in the network has several different paths to the other nodes, changing the position or removing a node of a mesh network simply forces the nodes to find the new best paths (thanks to an algorithm embedded in the network) to the other nodes.

In the 802.11s standard (an IEEE 802.11 amendment for mesh networking), the default routing protocol is called Hybrid Wireless Mesh Protocol (HWMP), a combination of AODV (Ad hoc On-Demand Distance Vector) and tree-based routing. This protocol provides both proactive and reactive path selection and must be supported in all mesh stations implementing 802.11s.
However, 802.11s standard allows vendors to operate using different routing protocols: in our experiments, in particular, we adopted Meshmerize, a wireless mesh routing protocol for dynamic networks.

2.2 Meshmerize

Popular mesh routing protocols, like B.A.T.M.A.N or AODV, use only single-path routing and therefore they are not optimized for dynamic scenarios, where nodes move around in the network and the best paths can change frequently. In this type of protocol, in fact, information sent between two nodes follow a predetermined best route (that can consist of multiple hops) and this mechanism only works well when the positions of the nodes are fixed and so in a static environment. On the other hand, when the nodes can freely move within the network boundaries, we could end up with frequent failures, since the best paths become invalid and have to be recalculated.

Meshmerize has been therefore developed to overcome the issue of the classical mesh routing protocols in dynamic environments. More precisely, Meshmerize is a wireless mesh routing protocol that, exploiting the broadcast nature of wireless medium, uses opportunistic multi-path routing to achieve high resilience. Multi-path routing means that, differently from the classical routing protocols, more than one node can transmit same information to the destination node. Since transmitting on multiple paths reduces the capacity of the links and, consequently, the maximum throughput of the network, Meshmerize protocol tries to find the ideal set of nodes that should relay the packet basing on routing metrics.

Using this mechanism, Meshmerize is able to achieve a very high resilience in a mesh network in case of time-variant links or link losses and is therefore perfectly suitable for a dynamic environment.

After this brief presentation on how wireless mesh network and Meshmerize protocol work, we will focus in the next section on our project, starting from the design part.

3 Design

For the system design, three essential problems had to be considered in order to reach the goals for the project.

- **What data is needed?**
- **How is data fetched from a client?**
- **How often should the data be fetched?**

3.1 What data is needed

From the Meshmerize evaluation kit documentation [2], two important commands can be used to gather useful information about a current node state in the mesh network. The first command `Meshmerize neighbor` returns all current neighbours of a node, including the success probability for both transmission & receiving, normalised to 255 to each neighbour node. This indicates how far the devices are from each other, or how good the signal is in their environment. The second command `Meshmerize Originator` returns the routing metrics for the current node. It shows the next hop, which is calculated as a metric value and includes decisions such as {Rate to destination, delivery probability, number of packets}. Additionally each data sample needs to be labelled with the current IP of the node and the current time, so that each sample can be uniquely identified.

3.2 How is data fetched from a client

In order for a client to analyse the data from the different nodes, the data needs to be available for the client. The given nodes have a constrained, stripped Linux based system installed, where the simplest form of moving data from one node to another is via the `scp` command. Instead of the client fetching data-logs from each of the different nodes in the network, we want to centralise all the different logs on the router, as this saves time and gives us a centralised data access point. As seen on Figure 4 the different nodes in the mesh network send their data via `scp` to the router, which acts as a centralised data station for the client. The client can then fetch all the data logs in one request from the router.

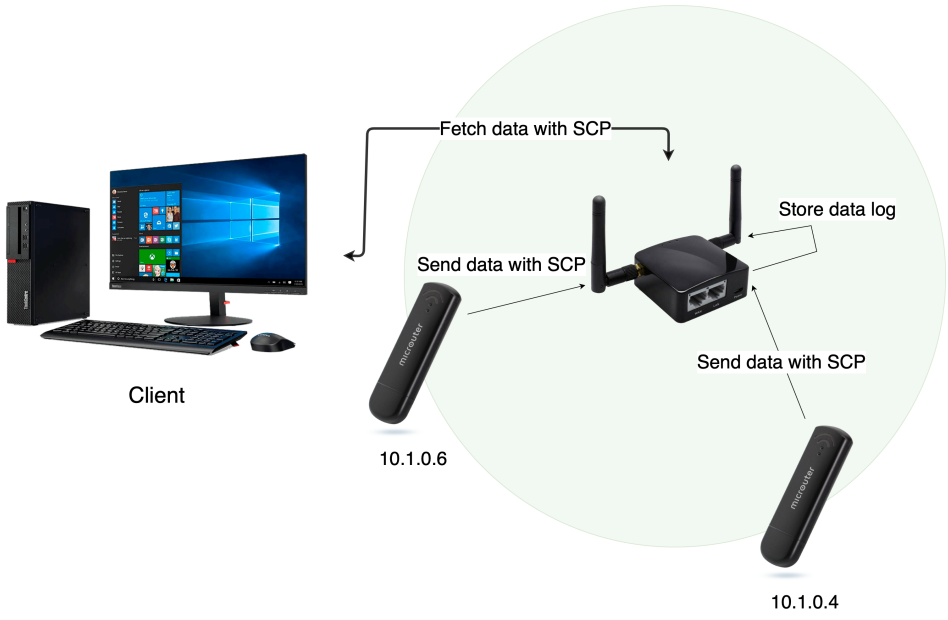


Figure 3: Fetching of data logs

3.3 How often should the data be fetched

From a architectural point of view, there are two different approaches to serve the client with data.

- **Bulking data logs:** Each node logs their data over an interval, generating a sequence of many samples. When a reasonable amount of data samples is generated these are pushed to the central data station (router). This approach restricts the client to wait for the data. When data is ready, the client will have more data to analyse and visualise.
- **Single data log:** Each node logs their data and, for each sample, the node immediately sends the single data sample to the central data station (router). This approach serves the client with live data, but only serves one sample to analyse and visualised.

The chosen approach is single data logging. This allows us to fetch the most resent data, and therefore gives us the possibility to plot live data.

4 Implementation

Reasoned that the nodes have a constrained stripped Linux system installed, it is chosen to implement logging of the different data and sending of the data to the central data station (router) in shell scripts (.sh). For the client implantation it is chosen to use `Python` reasoned the support for easy loading of data via `Pandas` and plotting via `Mathplotlib`.

4.1 Node implementation

The implementation of the mesh data logger script can be seen at figure 4. When started, the code runs for an infinite amount of time. Each data sample is logged in an interval of one second. In each data sample the time from the OS is fetched and formatted. The IP of device is fetched and via regex the specific IP for the mesh network is returned. At last, both results of commands `Mesherize neighbor` & `Meshmerize originator` are logged. After the log file is created, it gets pushed to the central data station (router), that serves the data-logs for the client.

meshlogger_04.sh

```
timestamp(){
 echo $(date "+%Y/%m/%d-%H:%M:%S") #current time
}

getIPAddr(){
 echo $(ifconfig | grep -E -o 'inet addr:10.1.0.\d')
}

counter=0
while true
        do
        sleep 1
        timevariable=$(timestamp)
        IPAddr=$(getIPAddr)
        counter=$(($counter+1))
        echo "IP.addr: " $IPAddr > logfile_node04.txt
        echo "Current timestamp: " $timevariable >> logfile_node04.txt
        echo "Number of file iterations: " $counter >> logfile_node04.txt
        echo "" >> logfile_node04.txt
        meshmerize neighbor >> logfile_node04.txt
        echo "" >> logfile_node04.txt
        meshmerize originator >> logfile_node04.txt

        scp logfile_node_04.txt root@10.1.0.1:/root/logfiles
done
```

Figure 4: meshlogger_04.sh implementation

For each of the nodes in the network, the scripts will nearly be the same. The node with IP "10.1.0.4" will generate the file "logfile_node04.txt" and the node with IP "10.1.0.6" will generate the file "logfile_node06.txt" and so on. For the router the script is a little different, reasoned that dirty reads can occur when the client tries the fetch the log file. If the client tries to fetch the log file while the router has not finished with the generation of the file, the client will pull a corrupt file. To overcome this issue, the router will create a intermediate file called "logfile_node01NR.txt", the "NR" marks the file as "NOT READY". When the file is ready, it will rename the file as "logfile_node01".

4.2 Client implementation

The python code at figure 5 create two types of objects. The first class is the `MeshDataFetcher`: it takes the IP and username of the central data station, which enables `scp`. When `.fetch_data_from_device()` is called on the fetcher, all data is pulled from the centralised data station (router) with IP "10.1.0.1". To enable the `Pandas` lib, the files the ".txt" log files needs to be in ".csv" format. The `MeshDataParser` handles this conversion. Furthermore, the client will pull data from the central data station with an interval of 1 sec.

main.py

```python
if __name__ == "__main__":
    router_ip = "10.1.0.1"
    router_username = "root"
    data_fetcher = MeshDataFetcher(router_ip, router_username)
    data_parser = MeshDataParser()

    while(1):
        time.sleep(1)

        data_fetcher.fetch_data_from_device()
        data_parser.parse_log_to_csv("logfiles/logfile_node01.txt")
        data_parser.parse_log_to_csv("logfiles/logfile_node06.txt")
        data_parser.parse_log_to_csv("logfiles/logfile_node04.txt")

        data_frame_1 = mesh_data_analyser.load_data_table("logfile_node01.csv")
        data_frame_4 = mesh_data_analyser.load_data_table("logfile_node04.csv")
        data_frame_6 = mesh_data_analyser.load_data_table("logfile_node06.csv")
```

Figure 5: Fetching data + parsing

5 Test setup

The test setup for the system can be seen in Figure 6. It shows a client hooking into to mesh network over LAN, which enables the client to fetch data from the central data station (router).

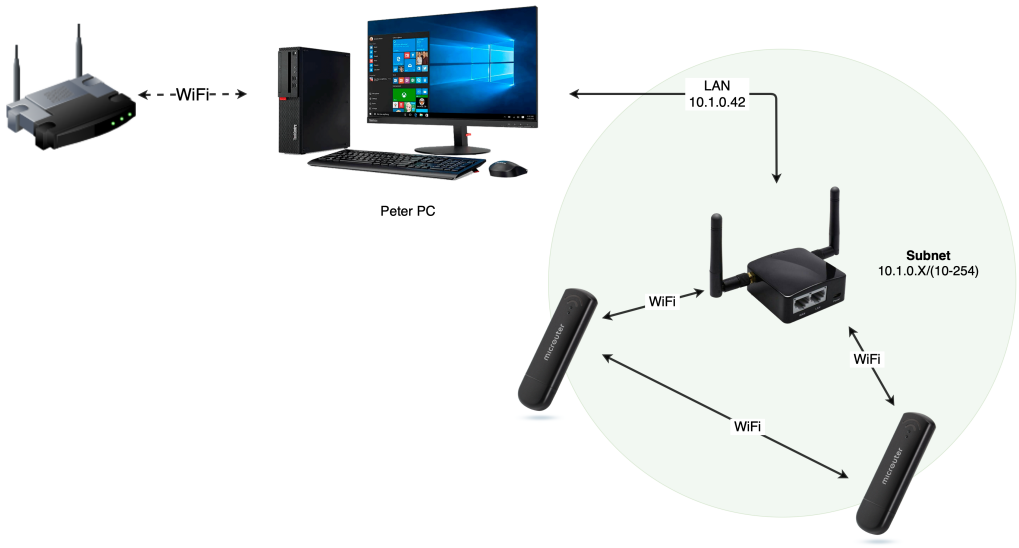


Figure 6: Test setup

5.1 Prerequisite nodes

For all of the nodes, the mesh data logger script must be deployed. This can be done using `scp`. Each of the scripts must also be up and running. This can be done by simply using `SSH` to get into each node, and starting the ".sh" script. As an alternative, these could also be started on boot, by configuring the OS to do so.

5.2 Prerequisite client

The client's IP must be configured to "10.1.0.X" where X is in the range of $10 - 254$. This enables the client to be on the same network as the nodes. For the client python code the following packages must be installed using install tool `pip`, see the below figure 7.

prereq

```
pip install tabulate
pip install networkx
pip install pandas
pip install matplotlib
pip install sympy
pip install numpy==1.19.3
```

Figure 7: Python prerequisite

6 Experiments

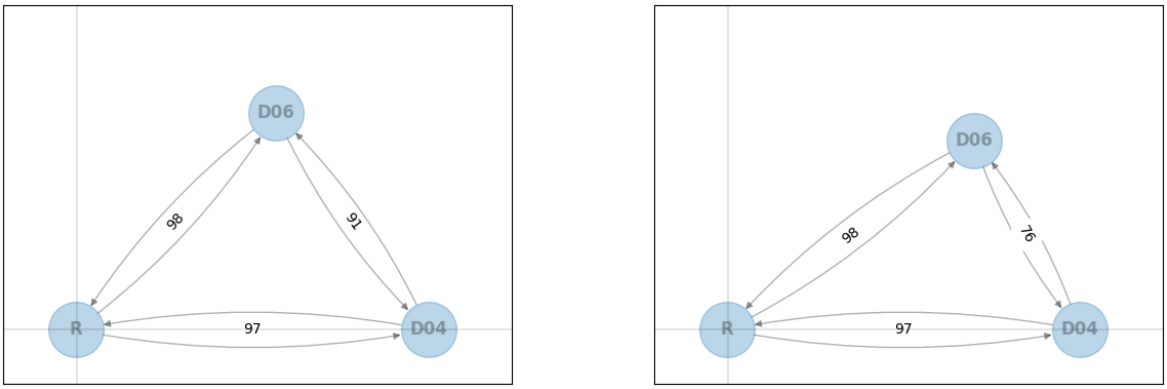
For the following section, we will try to document the reasoning for the three experiments and which results were obtained. The three are as follows:

1. Dynamically plotting nodes based on their distance (RX/TX) values.
2. Dynamically plotting the shortest route from root to destination node (based of metrics from the meshmerize commands).
3. Changing the topology, by moving nodes around.

6.1 Dynamically plotting nodes

As the first experiment, we had managed to centralise the data from the meshmerize originator and neighbour commands from the two dongles and router. We had a constant flow of data and wanted to see how the mesh network would dynamically rate the transfer (TX) and receive (RX) rates for the neighbour nodes. Could we plot the three nodes in an XY-coordinate system based on their RX/TX rates, we would be able to see the data visualised.

Learning how the distance matrix worked in conjunction with finding coordinates, a three node plot was created. This plot would constantly update itself based upon the received data from the nodes and therefore would dynamically plot continuously. Looking at figure 8, two images are shown in subplot 8a and 8b. These plots are derived from the same setup and code, but with a snapshot of a weaker RX/TX rate between dongle 06 and dongle 04.



(a) Dynamically plotted nodes 1

(b) Dynamically plotted nodes 2

Figure 8: Images of dynamically plotting nodes

6.2 Dynamically plotting shortest path

As the second experiment, we specifically wanted to visualise how the meshmerize WiFi network choose its shortest path. We had in the first experiment tried to visualise the meshmerize neighbour command, but now wanted to use the data from the meshmerize originator command. This data would provide us with the metric score for each nodes' neighbour, where the lower the number the more preferred the route.

At the time of this writing, the method finding the shortest path is based purely on a hardcoded solution from the received node data. In general we simply check for metric score and color the shortest path.
Looking at figure 9, we see the plotting of the shortest path from the router to the destination dongle 04.

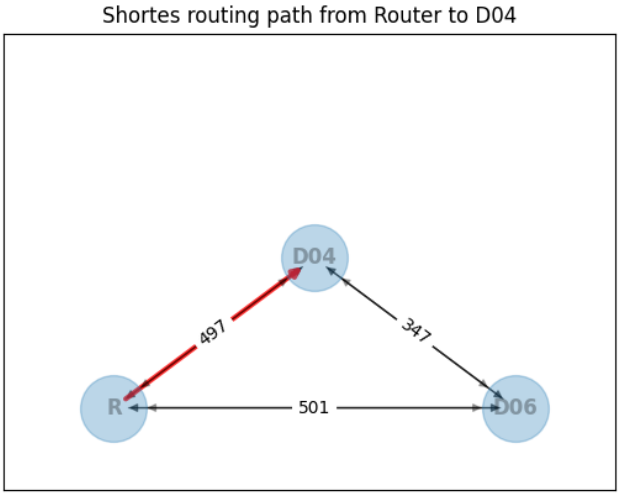
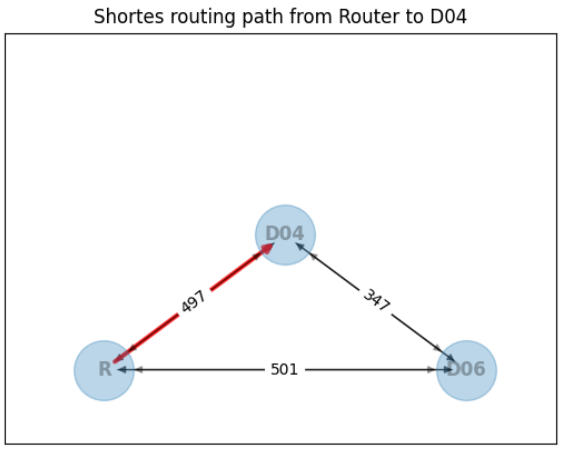


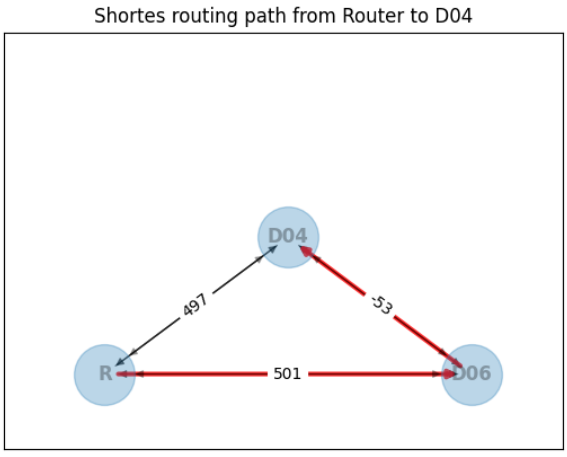
Figure 9: Dynamically plotted shortest path

6.3 Changing topology

For the last experiment, we wanted to try to change to topology for the nodes and see how the dynamic plotting of the edges would look like. The problem here was, that the base range of wifi is up to 100 meters. Having the setup located in one of our teams houses, we weren't able to split the nodes apart. To solve this problem we ended up modelling our own data and passing this as simulation flag in our plotting function. Having this option would allow us to see the shortest path dynamically change its preferred edges. Looking at figure 10, we see the two subplots 10a and 10b. The same data but with the simulation flag set to `True`, we see how the shortest path from router to the dongle 04 changes.



(a) Dynamically plotted shortest path 1



(b) Dynamically plotted shortest path 2

Figure 10: Images of dynamically plotting the shortest path

7 Conclusion

From the different experiments, it is possible to get a more visually and deep understanding of how the routing schemes behave in real time. Even though the experiments were performed in a environment where it was hard to change the topology, the simulation helped to show the expected topology change, when a new shorter path was found. Though the simulation results gave some decent re-

sults, the experiments should be retried in a better environment where the 100 meter range could be achieved to change to topology, instead of simulated data.

The built data gathering system, differently, has many areas to improve, and these should be taken into account for future developments on the project.

- **MeshDataParser:** Currently the MeshDataParser does not support the cases when links break or when new nodes arrive to the system. A more generic way of parsing these, could be to make any missing-data or broken links to have `NaN`-values. This enables the network graph to update its nodes and edges properly.
- **MeshDataAnalyser:** Currently only supports 3 nodes. Calculation of the shortest path and plotting does not support arrival of new nodes. This could be supported by adding `NaN`-values in the parser.

Lastly, one could argue that comparing the standard WiFi 802.11 protocol vs. the mesh 802.11s protocol could add some value. In terms of the performance, we haven't been able to conclude which protocol yields a faster packet transfer speed, since we didn't have enough time.

8 Future work

The 802.11s mesh routing protocol has and is already implemented in various systems around the world. Allowing wireless network architectures of multiple nodes to still intercommunicate to an otherwise out-of-range system is a great characteristic that many system benefit from. For the specific young startup Meshmerize, they've already started to provide reliable connectivity between devices, even in the absence of an infrastructure [1]. Through drone swarms, agriculture, mining/construction or emergency services, we as a group would argue that the expanded 802.11s meshmerize protocol, with some more "word spreading", will be explored and used upon in the coming future.

9 Bibliography

Websites

[1] Meshmerize. *Meshmerize*. URL: https://www.meshmerize.net/.

[3] Wikipedia. *IEEE 802.11s*. URL: https://en.wikipedia.org/wiki/IEEE_802.11s.

[4] Wikipedia. *Wireless ad hoc network*. URL: https://en.wikipedia.org/wiki/Wireless_ad_hoc_network.

[5] Wikipedia. *Wireless mesh network*. URL: https://en.wikipedia.org/wiki/Wireless_mesh_network.