



Master of Applied Computer Science

Embedded Connectivity

Summer Term 2021

Paper work

Vipin Thomas: 00812684

Majd Hafiri: 00812127

Date 09-07-2021

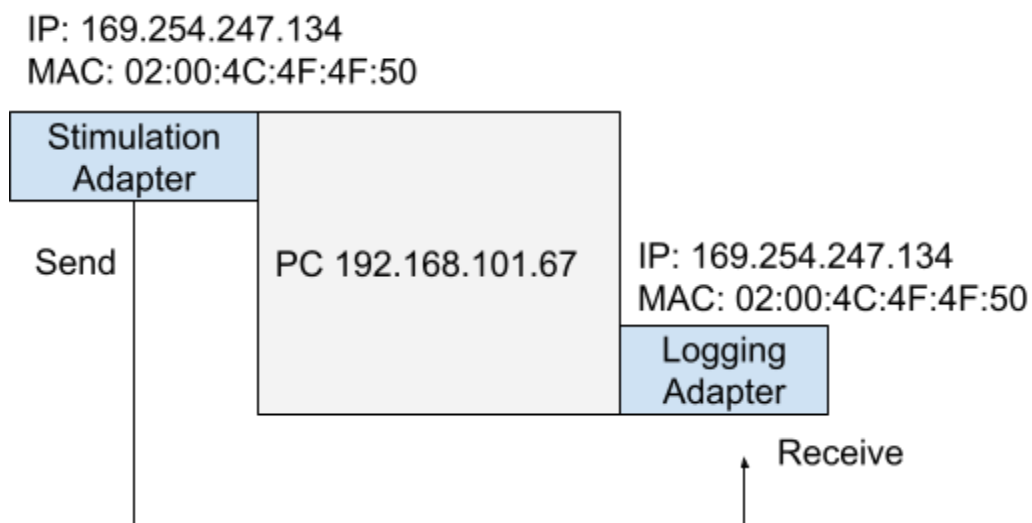
Table of contents

PART1: Startup with ANDI-tool and Loopback	3
• Layer 2 frames	4
- Send script.....	4
- Receive Script	5
• Layer 3 packets	7
- Send script.....	7
- Receive Script.....	8
 PART 2 : communication path over MediaGateway	11
• Send script.....	12
• Receive Script.....	12
• simple connection and VLAN	14
 PART 3: Integration into a network structure	15
• Media Gateway configurations	16
 Part 4 - Project: ANDi Graphical User Interface	22
• Introduction	22
• Main features.....	23
• GUI Controls.....	23
• Implementation	25
 Challenges	31
Work Distribution	32
Conclusion	33
References	34

PART 1: Startup with ANDi tool and Loopback

The main goal in this part is to send Layer 2 frames and layer 3 packets from and to the same machine which is called a Loopback connection. Four python scripts were written in the ANDi tool. Two scripts for the send and receive functionalities for Layer 2 frames, and the other Two scripts for the same purpose for Layer 3 packets.

We have first set the Adapter Configurations in ANDi tool by changing the Physical network adapter for Stimulation and Logging to Loopback adapters (Ethernet 3) Which are used to send and Receive frames and packets respectively. The following figure illustrates the loopback connection:



1.1- Layer 2 frames:

Send script

First of all, we set a source mac address to the ethernet message as follows:

```
custom_mac="11:22:33:44:55:66"  
g_ethernet_msg.mac_address_source = custom_mac
```

and since it's a loopback connection, the source mac address doesn't have a major importance. After that we set the destination mac address of the ethernet message to the mac address of Loopback adapter of the same machine:

```
g_ethernet_msg.mac_address_destination = "02:00:4C:4F:4F:50"
```

In this script , we're sending 20 frames and each frame is sent after one second. This happens in a for loop with a sleep time of 1 second, and before the frame is sent , we change the first byte of the dataset by increasing it by 1 which helps in the verification step. After preparing the dataset, we add it to the payload of the ethernet message and then send it

```
for i in range(20):  
    if i < 10:  
        dataset = "0{0} 02 03 04 09".format(i)  
    else:  
        dataset = "{0} 02 03 04 09".format(i)  
    g_ethernet_msg.payload = System.Array[Byte](bytearray.fromhex(dataset))  
    g_ethernet_msg.send()  
  
    sleep(1)
```

Receive script

There's a function which gets called when the machine captures incoming messages , this function takes the message received as a parameter and then it checks if the source mac address in the message matches the custom mac address which is previously defined the same as in the send script and it prints out the message on the ANDi tool console. Knowing that we can also check if the destination mac address matches with the loopback adapter mac address.

```
def on_eth_msg_received(msg):  
  
    if msg.mac_address_source == custom_mac:  
        print "Received following message from {0}: {1}".format(custom_mac,  
msg)
```

In order to capture the 20 messages sent from the machine , we used the following lines of code with a time sleep of 20 seconds.

```
g_ethernet_msg.on_message_received += on_eth_msg_received  
g_ethernet_msg.start_capture()  
sleep(20)  
g_ethernet_msg.stop_capture()  
g_ethernet_msg.on_message_received -= on_eth_msg_received
```

After running the two scripts , the receive script will print out the received message information as follows:

ANDi Console output

```
Scripts Output

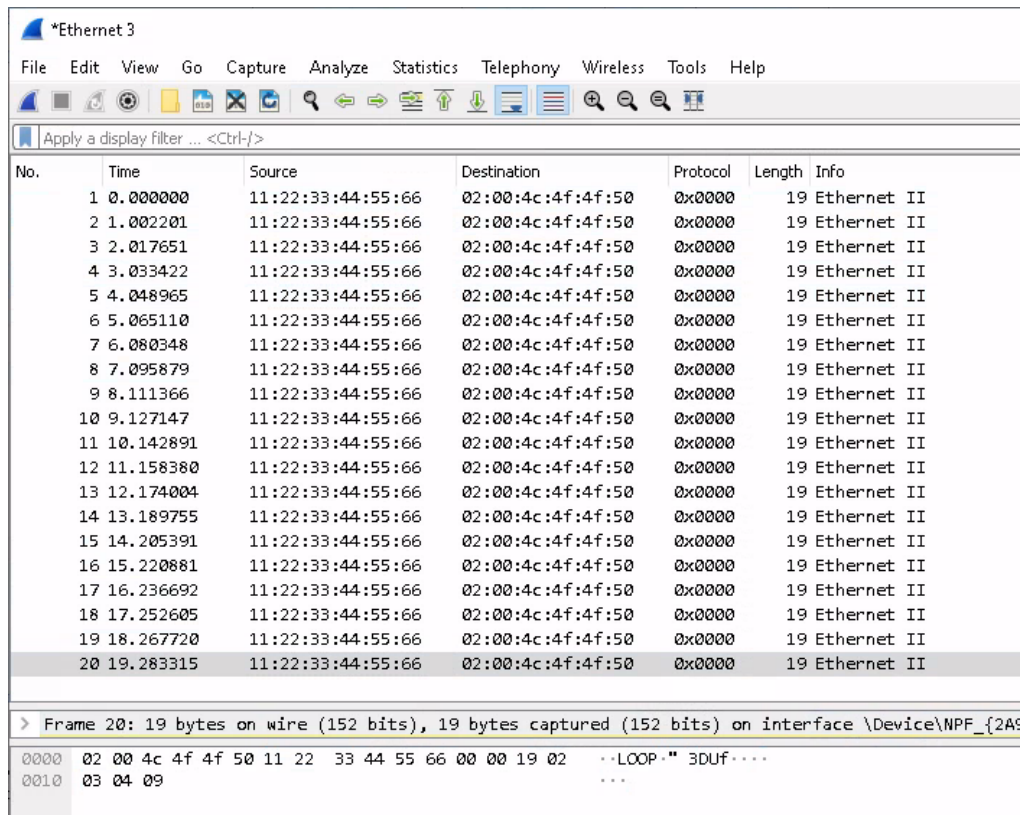
[2021-07-04 13:29:57]: Start 'Exercisel_layer2_receive'
[2021-07-04 13:30:00]: Start 'Exercisel_l1ayer2_send'
[2021-07-04 13:30:00]: Received following message from 11:22:33:44:55:66 Mac src: 11:22:33:44:55:66 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown
[2021-07-04 13:30:00]: Payload Data : {02349}
[2021-07-04 13:30:01]: Received following message from 11:22:33:44:55:66 Mac src: 11:22:33:44:55:66 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown
[2021-07-04 13:30:01]: Payload Data : {12349}
[2021-07-04 13:30:02]: Received following message from 11:22:33:44:55:66 Mac src: 11:22:33:44:55:66 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown
[2021-07-04 13:30:02]: Payload Data : {22349}
[2021-07-04 13:30:03]: Received following message from 11:22:33:44:55:66 Mac src: 11:22:33:44:55:66 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown
[2021-07-04 13:30:03]: Payload Data : {32349}
[2021-07-04 13:30:04]: Received following message from 11:22:33:44:55:66 Mac src: 11:22:33:44:55:66 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown
[2021-07-04 13:30:04]: Payload Data : {42349}
[2021-07-04 13:30:05]: Received following message from 11:22:33:44:55:66 Mac src: 11:22:33:44:55:66 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown
[2021-07-04 13:30:05]: Payload Data : {52349}
[2021-07-04 13:30:06]: Received following message from 11:22:33:44:55:66 Mac src: 11:22:33:44:55:66 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown
[2021-07-04 13:30:06]: Payload Data : {62349}
[2021-07-04 13:30:07]: Received following message from 11:22:33:44:55:66 Mac src: 11:22:33:44:55:66 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown
[2021-07-04 13:30:07]: Payload Data : {72349}
```

We have also used ANDi traffic viewer and Wireshark tools to verify that the frames reached the destination.

ANDi traffic viewer:

Source	Destination	Protocol	Length	Info	Payload
11:22:33:44:55:66	02:00:4c:4f:4f:50	Ethernet	19		00 02 03 04 09
11:22:33:44:55:66	02:00:4c:4f:4f:50	Ethernet	19		01 02 03 04 09
11:22:33:44:55:66	02:00:4c:4f:4f:50	Ethernet	19		02 02 03 04 09
11:22:33:44:55:66	02:00:4c:4f:4f:50	Ethernet	19		03 02 03 04 09
11:22:33:44:55:66	02:00:4c:4f:4f:50	Ethernet	19		04 02 03 04 09
11:22:33:44:55:66	02:00:4c:4f:4f:50	Ethernet	19		05 02 03 04 09

Wireshark:



The image shows a Wireshark capture of 20 Ethernet II frames. The packet list table is as follows:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
2	1.002201	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
3	2.017651	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
4	3.033422	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
5	4.048965	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
6	5.065110	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
7	6.080348	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
8	7.095879	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
9	8.111366	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
10	9.127147	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
11	10.142891	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
12	11.158380	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
13	12.174004	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
14	13.189755	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
15	14.205391	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
16	15.220881	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
17	16.236692	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
18	17.252605	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
19	18.267720	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II
20	19.283315	11:22:33:44:55:66	02:00:4c:4f:4f:50	0x0000	19	Ethernet II

Below the packet list, the details of Frame 20 are shown:

> Frame 20: 19 bytes on wire (152 bits), 19 bytes captured (152 bits) on interface \Device\NPF_{2A...}

Offset	Hex	ASCII
0000	02 00 4c 4f 4f 50 11 22 33 44 55 66 00 00 19 02	..LOOP.. 3DUF....
0010	03 04 09	...

1.2- Layer 3 packets

In this part , two more scripts were used to send and receive IP packets on the same machine. and there's only a slight difference from the scripts which were previously mentioned.

Send script

First of all we create a UDP message which will be sent to the receiver(Same machine) as follows:

```
udpMessage = message_builder.create_udp_message()
```

This UDP message must have the IP source and destination addresses, and in this case since we're still working with the loopback connection , the IP

source and destination addresses are equal to the IP addresses of the Loopback adapters (Stimulation and Logging):

```
udpMessage.ipv4_header.ip_address_source = "169.254.247.134"  
udpMessage.ipv4_header.ip_address_destination = "169.254.247.134"
```

This udp message should also include the source port and a destination port and they are set as the following:

```
udpMessage.udp_header.port_source = 9999  
udpMessage.udp_header.port_destination = 10000
```

Finally , the exact same piece of code used in the send script for Layer 2 frames was used to send 1 IP packet every one second for 20 seconds and change the first byte of the dataset in the payload for better verification.

Receive script:

It's almost the same as the receive script for layer 2 frame but only few changes needed. We first create two strings variables which hold the source and destination IP addresses and they're set to the Loopback adapter IP address:

```
destination_ip = "169.254.247.134"  
source_ip = "169.254.247.134"
```

And In function on_eth_msg_received(msg) we check if the destination IP address is equal to the loopback adapter IP address:

```
If msg.get_ipv4_layer().ipv4_header.ip_address_destination ==  
destination_ip:
```


After running both script , the receive function will print the received message on ANDi console:

ANDi Console output

```
[2021-07-04 13:32:30]: Start 'Exercise1_layer3_receive'
[2021-07-04 13:32:33]: Start 'Exercise1_layer3_send'
[2021-07-04 13:32:33]: Received following message from 169.254.247.134: Mac src: 02:00:4C:4F:50 Mac dest: FF:FF:FF:FF:FF:FF EtherType: IPv4
[2021-07-04 13:32:33]: Payload Data : {6900330000321787194169254247134169254247134391539160139816402349}
[2021-07-04 13:32:34]: Received following message from 169.254.247.134: Mac src: 02:00:4C:4F:50 Mac dest: FF:FF:FF:FF:FF:FF EtherType: IPv4
[2021-07-04 13:32:34]: Payload Data : {6900330000321787194169254247134169254247134391539160139716412349}
[2021-07-04 13:32:35]: Received following message from 169.254.247.134: Mac src: 02:00:4C:4F:50 Mac dest: FF:FF:FF:FF:FF:FF EtherType: IPv4
[2021-07-04 13:32:35]: Payload Data : {6900330000321787194169254247134169254247134391539160139616422349}
[2021-07-04 13:32:36]: Received following message from 169.254.247.134: Mac src: 02:00:4C:4F:50 Mac dest: FF:FF:FF:FF:FF:FF EtherType: IPv4
[2021-07-04 13:32:36]: Payload Data : {6900330000321787194169254247134169254247134391539160139516432349}
[2021-07-04 13:32:37]: Received following message from 169.254.247.134: Mac src: 02:00:4C:4F:50 Mac dest: FF:FF:FF:FF:FF:FF EtherType: IPv4
[2021-07-04 13:32:37]: Payload Data : {6900330000321787194169254247134169254247134391539160139416442349}
[2021-07-04 13:32:38]: Received following message from 169.254.247.134: Mac src: 02:00:4C:4F:50 Mac dest: FF:FF:FF:FF:FF:FF EtherType: IPv4
[2021-07-04 13:32:38]: Payload Data : {6900330000321787194169254247134169254247134391539160139316452349}
[2021-07-04 13:32:39]: Received following message from 169.254.247.134: Mac src: 02:00:4C:4F:50 Mac dest: FF:FF:FF:FF:FF:FF EtherType: IPv4
[2021-07-04 13:32:39]: Payload Data : {6900330000321787194169254247134169254247134391539160139216462349}
[2021-07-04 13:32:40]: Received following message from 169.254.247.134: Mac src: 02:00:4C:4F:50 Mac dest: FF:FF:FF:FF:FF:FF EtherType: IPv4
[2021-07-04 13:32:40]: Payload Data : {6900330000321787194169254247134169254247134391539160139116472349}
[2021-07-04 13:32:41]: Received following message from 169.254.247.134: Mac src: 02:00:4C:4F:50 Mac dest: FF:FF:FF:FF:FF:FF EtherType: IPv4
[2021-07-04 13:32:41]: Payload Data : {6900330000321787194169254247134169254247134391539160139016482349}
```

Besides, we have also used ANDi traffic viewer and Wireshark tools to verify that the packets reached the destination.

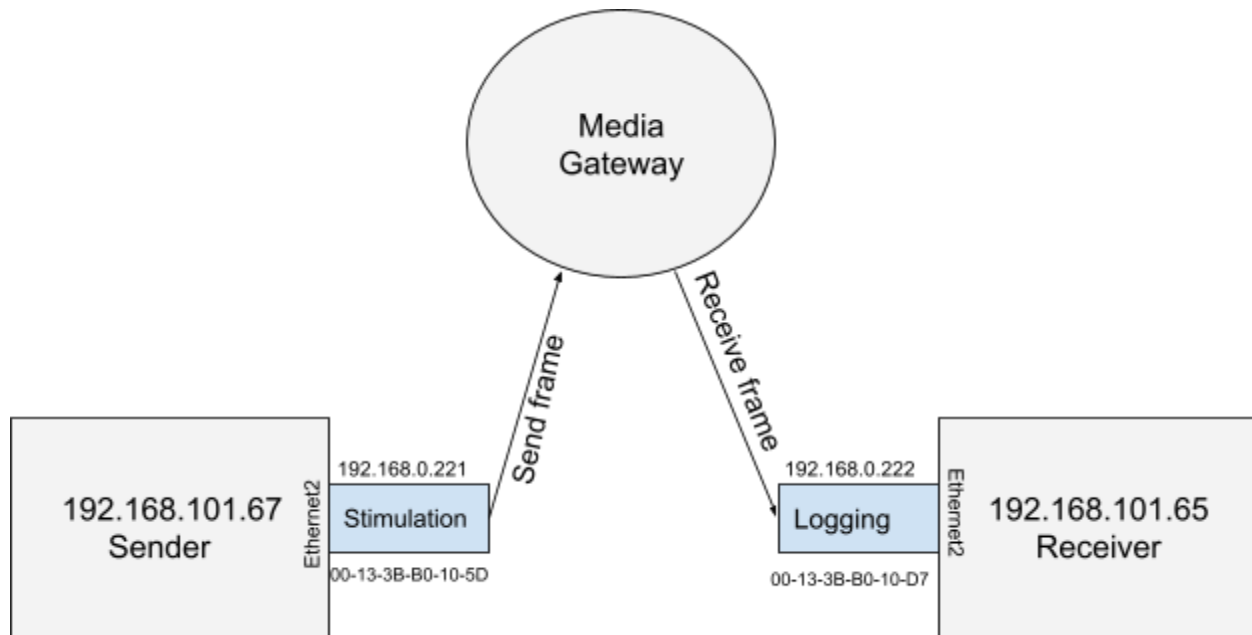
ANDi traffic viewer

Filter : <All Messages>										
No	Time	Delta	Vlan	Direction	Bus	Id	Source	Destination	Protocol	Length
10	20.203197	1.015716			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
11	21.219315	1.016118			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
12	22.234349	1.015034			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
13	23.250060	1.015711			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
14	24.265766	1.015706			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
15	25.281299	1.015533			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
16	26.296904	1.015605			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
17	27.312497	1.015593			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
18	28.328135	1.015638			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
19	29.343796	1.015661			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
20	29.390803	0.047007			Ethernet		02:00:4c:4f:4f:50, 0.0.0.0	ff:ff:ff:ff:ff:ff, 255.255.255.255	DHCP	342
21	30.359488	0.968685			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
22	31.375021	1.015533			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
23	32.390598	1.015577			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
24	33.406184	1.015586			Ethernet		02:00:4c:4f:4f:50, 169.254.247.134	ff:ff:ff:ff:ff:ff, 169.254.247.134	UDP	47
> General Information										
> Ethernet: Ethernet II, dst: FF:FF:FF:FF:FF:FF (Broadcast), src: 02:00:4C:4F:4F:50 (Unicast Address, Vendor Code Not Available)										
> IPv4: Internet Protocol Version 4 , src: 169.254.247.134 , dst: 169.254.247.134										
> UDP: User Datagram Protocol, srcport: 9999, dstport: 10000										
▼ Data: [5 Bytes]										
Data: 19 02 03 04 09										

Wireshark:

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help							
Apply a display filter ... <Ctrl-/>							
No.	Time	Source	Destination	Protocol	Length	Info	
5	17.608297	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
6	18.624389	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
7	19.639768	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
8	20.655429	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
9	21.670981	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
10	22.686631	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
11	23.702347	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
12	24.718465	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
13	25.733499	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
14	26.749210	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
15	27.764916	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
16	28.780449	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
17	29.796054	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
18	30.811647	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
19	31.827285	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
20	32.842946	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
21	32.889953	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0x...	
22	33.858638	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
23	34.874171	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
24	35.889748	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5
25	36.905334	169.254.247.134	169.254.247.134	UDP	47	9999 → 10000	Len=5

PART 2 : communication path over MediaGateway



In this part , we established a real point to point connection between the two computers on workstation 1 which are connected to the same media gateway, and were able to send layer 2 frames from one computer and receive them on the other computer.

Device	Connection	MAC	IP	application/usecase
computer 1	Ethernet	80-89-A5-ED-B9-7F	192:168.101.67	Internet
	Ethernet 1	00-13-3B-B0-10-5E	192:168.0.201	Configuration of MediaGate-way
	Ethernet 2	00-13-3B-B0-10-5D	192:168.0.221	Connected over MediaCon-verter to Mediagateway
computer 2	Ethernet	80-89-A5-F2-BF-6B	192:168.101.65	Internet
	Ethernet 1	00-13-3B-B0-10-D8	192:168.0.202	Connected to MediaGateway e.g. for Wireshark
	Ethernet 2	00-13-3B-B0-10-D7	192:168.0.222	Connected over MediaCon-verter to MediaGateway
WebCam		00-62-6E-93-52-66	192.168.0.171	Connected to MediaGateway
Media Gateway		70-B3-D5-28-DF-AC	192.168.0.49	

First of all we have set the Logging and Stimulation Adapters on ANDi tool for both computers to Ethernet 2. The Stimulation adapter is used for transmission and the Logging adapter is used for receiving.

Send script

As shown in the figure above , we have used the Machine 192.168.101.67 as the sender. Therefore, in the send script, we set the source mac address of the message to the mac address of the sender machine by getting the mac address of the Stimulation (sender) adapter:

```
g_ethernet_msg.mac_address_source = Stimulation.get_mac()
```

And we also set the destination mac address of the message to the other machine's MAC Address (receiver) for Ethernet 2:

```
g_ethernet_msg.mac_address_destination = "00:13:3B:B0:10:D7"
```

Regarding the rest of the code, It remains the same as what we used in the send scripts mentioned earlier.

Receive Script

There's a very slight difference from the previous received scripts that were mentioned above. Inside the `on_eth_message_received` function , we check if the destination mac address of the message matches with the Logging Adapter of the receiver machine (Ethernet 2) to print the message on the ANDi console.

```
if msg.mac_address_destination == Logging.get_mac():
```

After running the two scripts, the receive script prints out the received messages on *ANDi console*:

ANDi traffic viewer

[illegible][illegible]

2.2 - simple connection and VLAN

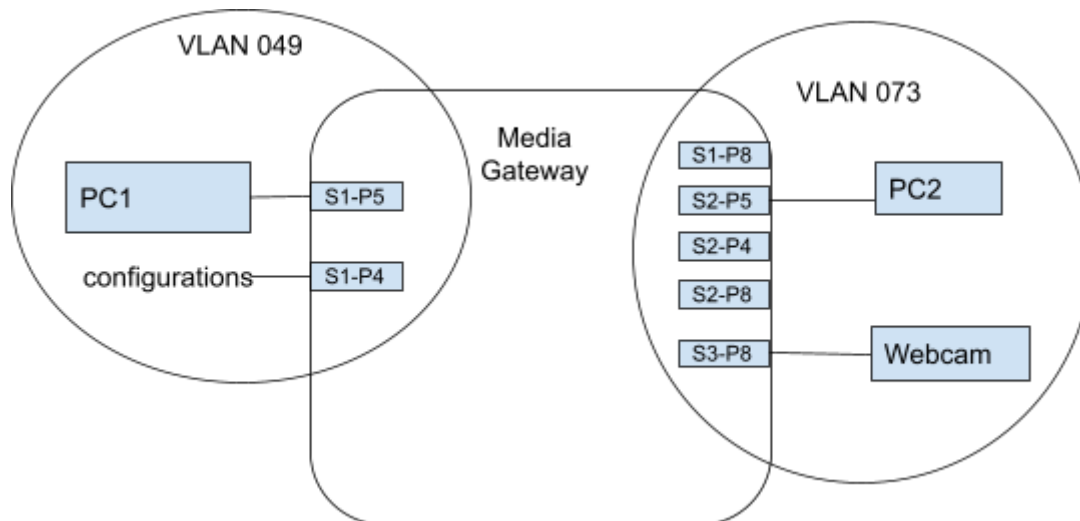
In this part , we have established a connection between two computers, with two VLANs, by making changes to the switch 1 vlan tag settings, ie, Switch will be part of both the vlans. . The first computer with the MAC address 00-13-3B-B0-10-5D is connected to port 5 of switch 1 and the second computer with the MAC address 00-13-3B-B0-10-D7 is connected to port 5 of switch 2.

Firstly, We have enabled the vlan mode on the media gateway and then applied the following configurations for ports 4 and 5 of switch 1 in which the pc connected to this port is only allowed to configure the media gateway:

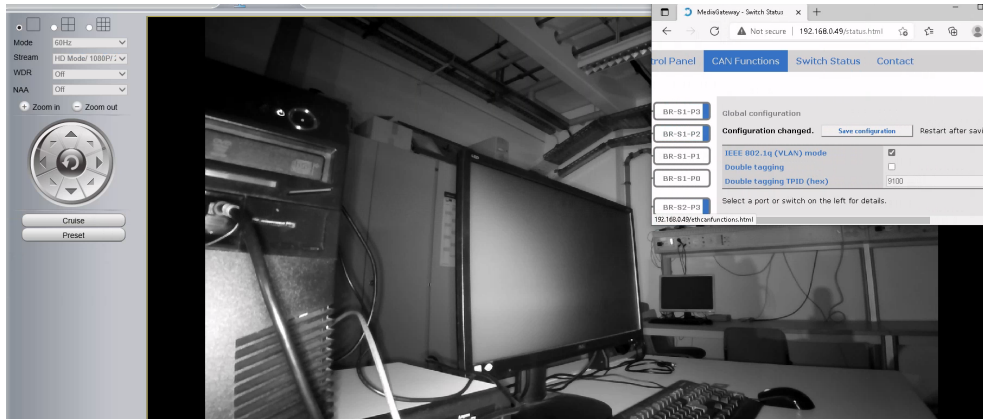
Default VLAN ID (hex)	049
VLAN membership	049
VLANs to untag	049

After that , we have set the membership of S1-P3, S1-P8, S2-P4,S3-P4 to 073 and to be able to connect to the webcam connected to S3-P8 , we have given the id 073 to the VLAN ID,membership and VLANs to untag. In this case the second PC which is connected to S2-P5 won't be able to configure the media gateway and both computers are connected to each other as well as the webcam.

The following figure shows how the devices are in different VLANs:

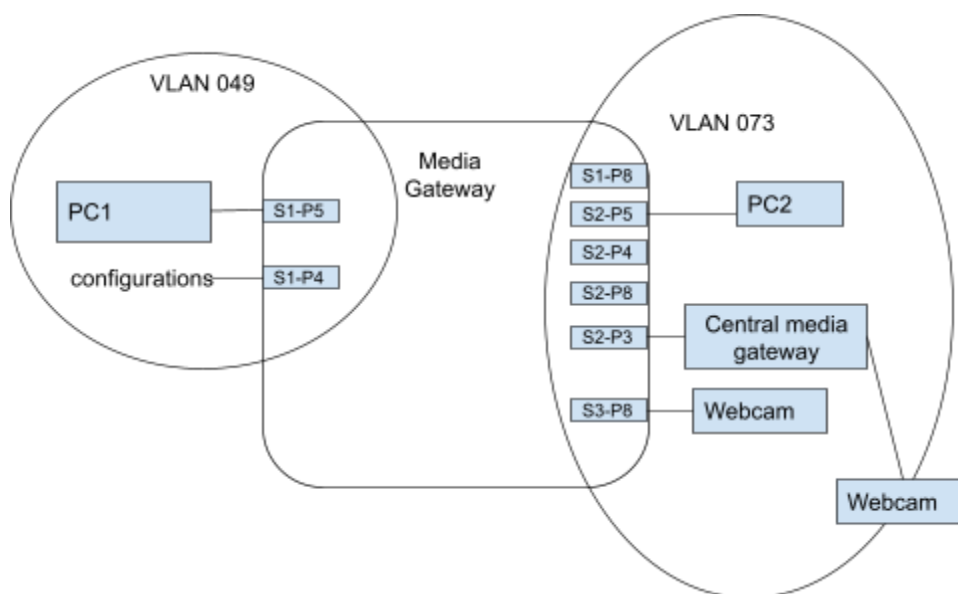


The traffic was verified by Wireshark between the two computers , and we also connected one of the computers to the webcam:



PART 3: Integration into a network structure

In this part , We have established a connection with the central media gateway via VLANs, the media gateway of Workstation 1 belongs to VLAN with id 049 and the central media gateway belongs to VLAN with id 073. Additionally We were able to access the camera connected to the central media gateway as well as sending and receiving frames between the two computers with the new configurations of the media gateway.



Media Gateway configurations:

1- Setting VLAN ID, VLAN membership and VLAN to untag to 049 for Ports 4 and 5 of switch 1.

Switch 1 Port 4	
Port name	S1-P4
Default VLAN ID (hex)	049
VLAN membership	049
VLANs to untag	049

Switch 1 Ethernet Port 5	
Port name	S1-P5
Default VLAN ID (hex)	049
VLAN membership	049
VLANs to untag	049
Egress VID remarking	Inner: As received ▼ Outer: As received ▼
Tx octets	1648296
Rx octets	1354781

2- Setting VLAN membership for S1-P8, S2-P8,S2-P4 and S3-P4 to 073 (same as the central gateway VLAN). Which will make these port members of the VLAN 073:

Switch 1 Port 8	
Port name	S1-P8
Default VLAN ID (hex)	
VLAN membership	073
VLANs to untag	
Egress VID remarking	Inner: As received ▼ Outer: As received ▼
Tx octets	540032
Rx octets	14271236

Switch 2 Port 8	
Port name	S2-P8
Default VLAN ID (hex)	
VLAN membership	073
VLANs to untag	
Egress VID remarking	Inner: <input type="text" value="As received"/> Outer: <input type="text" value="As received"/>
Tx octets	1313127
Rx octets	2367347

Switch 2 Port 4	
Port name	S2-P4
Default VLAN ID (hex)	
VLAN membership	073
VLANs to untag	
Egress VID remarking	Inner: <input type="text" value="As received"/> Outer: <input type="text" value="As received"/>
Tx octets	14532584
Rx octets	547974

Switch 3 Port 4	
Port name	S3-P4
Default VLAN ID (hex)	
VLAN membership	073
VLANs to untag	
Egress VID remarking	Inner: <input type="text" value="As received"/> Outer: <input type="text" value="As received"/>
Tx octets	2367551
Rx octets	1314032

3- Central media gateway is connected to Port 3 of switch 2, therefore we set the VLAN id and the membership to 073

Switch 2 BroadR-Reach® Port 3

Port name	BR-S2-P3
Default VLAN ID (hex)	073
VLAN membership	073
VLANs to untag	
Egress VID remarking	Inner: As received ▼ Outer: As received ▼
Tx octets	730670
Rx octets	14082141
Prevent Sleep Port	<input checked="" type="checkbox"/>
Enable port	<input checked="" type="checkbox"/>
BroadR-Reach® mode	Master ▼
Output level	Fullout ▼

4- configuring port 5 of switch 2 in which the second PC is connected to.

Switch 2 Ethernet Port 5

Port name	S2-P5
Default VLAN ID (hex)	073
VLAN membership	073
VLANs to untag	073
Egress VID remarking	Inner: As received ▼ Outer: As received ▼
Tx octets	3604460
Rx octets	146395
Prevent Sleep Port	<input checked="" type="checkbox"/>
Mirroring	P0 <input type="checkbox"/> P1 <input type="checkbox"/> P2 <input type="checkbox"/> P3 <input type="checkbox"/> P4 <input checked="" type="checkbox"/> P8 <input checked="" type="checkbox"/>

5- configuring port 8 of switch 3 by giving it VLAN id, membership and VLAN to untag of 073

Switch 3 Ethernet Port 8

Port name	S3-P8
Default VLAN ID (hex)	073
VLAN membership	073
VLANs to untag	073
Egress VID remarking	Inner: As received ▼ Outer: As received ▼

In the following Address Resolution tables for the three switches , we can see that there is a connection established between the two computers, Webcam of workstation 1 , and the webcam of the central media gateway.

Address Resolution Table - Switch 1

Address Resolution Table					Export
MAC address	VLAN ID	Fwd port	Age bit	Static bit	
00:62:6E:93:52:60	073	8	1	0	
00:13:3B:80:10:D7	073	2	1	0	
00:62:6E:93:52:66	073	8	1	0	
00:13:3B:80:10:D8	073	8	1	0	
70:B3:D5:28:DF:AC	049	4	1	0	
00:13:3B:80:10:80	073	8	1	0	
00:13:3B:80:10:4E	073	8	1	0	
00:13:3B:80:10:5D	073	3	1	0	
00:13:3B:80:10:5E	049	5	1	0	
00:02:D1:06:80:4A	073	8	1	0	

Address Resolution Table - Switch 2

Address Resolution Table					Export
MAC address	VLAN ID	Fwd port	Age bit	Static bit	
00:62:6E:93:52:60	073	3	1	0	
00:13:3B:80:10:D7	073	4	1	0	
00:62:6E:93:52:66	073	8	1	0	
00:13:3B:80:10:D8	073	5	1	0	
00:13:3B:80:10:80	073	3	1	0	
00:13:3B:80:10:4E	073	3	1	0	
00:13:3B:80:10:5D	073	4	1	0	
00:02:D1:06:80:4A	073	3	1	0	

Address Resolution Table - Switch 3

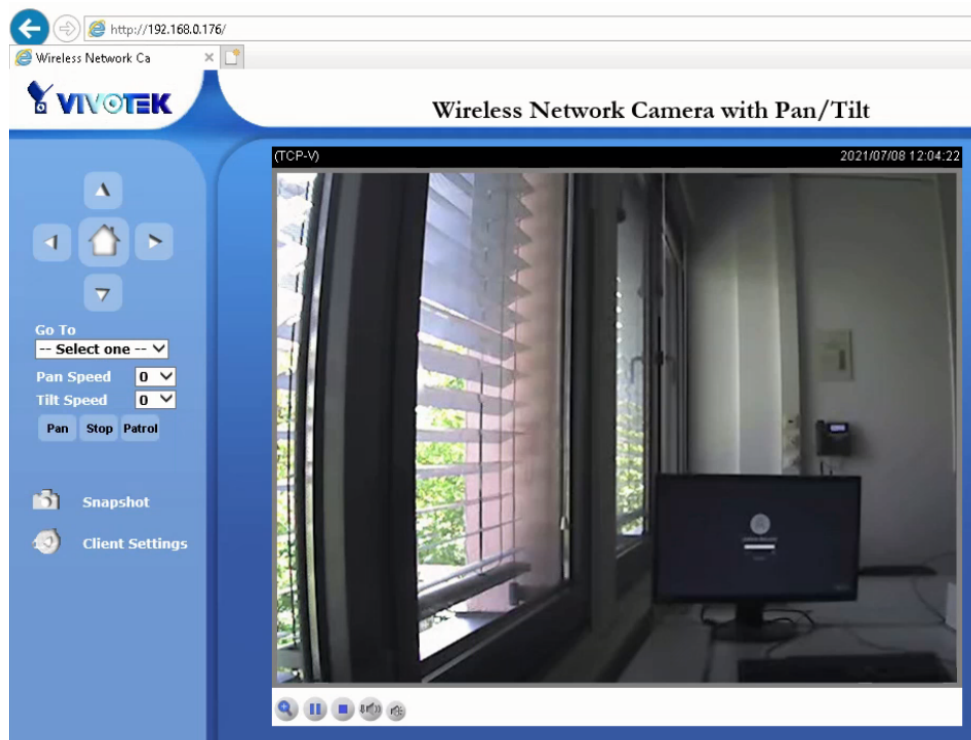
Address Resolution Table					Export
MAC address	VLAN ID	Fwd port	Age bit	Static bit	
00:62:6E:93:52:60	073	4	0	0	
00:13:3B:80:10:D7	073	4	1	0	
00:62:6E:93:52:66	073	8	1	0	
00:13:3B:80:10:D8	073	4	1	0	
00:13:3B:80:10:80	073	4	1	0	
00:13:3B:80:10:4E	073	4	1	0	
00:13:3B:80:10:5D	073	4	1	0	
00:02:D1:06:80:4A	073	4	0	0	

Central Webcam View:

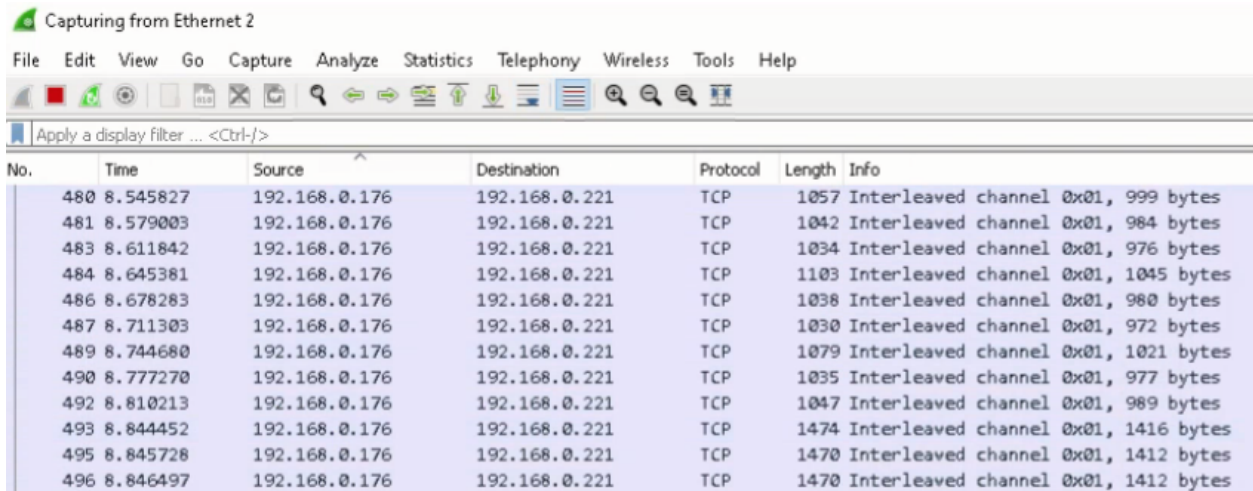
After configuring the media gateway and establishing connection with the central media gateway , we could connect to the central webcam using Internet Explorer and the Webcam IP: 192.168.0.176

User: maiec

Password: maiec20!



Moreover, the traffic was also analyzed by *wireshark tool*:



No.	Time	Source	Destination	Protocol	Length	Info
480	8.545827	192.168.0.176	192.168.0.221	TCP	1057	Interleaved channel 0x01, 999 bytes
481	8.579003	192.168.0.176	192.168.0.221	TCP	1042	Interleaved channel 0x01, 984 bytes
483	8.611842	192.168.0.176	192.168.0.221	TCP	1034	Interleaved channel 0x01, 976 bytes
484	8.645381	192.168.0.176	192.168.0.221	TCP	1103	Interleaved channel 0x01, 1045 bytes
486	8.678283	192.168.0.176	192.168.0.221	TCP	1038	Interleaved channel 0x01, 980 bytes
487	8.711303	192.168.0.176	192.168.0.221	TCP	1030	Interleaved channel 0x01, 972 bytes
489	8.744680	192.168.0.176	192.168.0.221	TCP	1079	Interleaved channel 0x01, 1021 bytes
490	8.777270	192.168.0.176	192.168.0.221	TCP	1035	Interleaved channel 0x01, 977 bytes
492	8.810213	192.168.0.176	192.168.0.221	TCP	1047	Interleaved channel 0x01, 989 bytes
493	8.844452	192.168.0.176	192.168.0.221	TCP	1474	Interleaved channel 0x01, 1416 bytes
495	8.845728	192.168.0.176	192.168.0.221	TCP	1470	Interleaved channel 0x01, 1412 bytes
496	8.846497	192.168.0.176	192.168.0.221	TCP	1470	Interleaved channel 0x01, 1412 bytes

Lastly, we were also able to send and receive layer 2 frames between the two computers using the send the receive script mentioned earlier(section..). And the traffic was analyzed by wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
4	1.004545	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
13	2.008399	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
35	19.979874	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
36	20.993463	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
46	21.995962	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
51	23.002380	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
53	24.016422	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
55	25.017608	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
57	26.026126	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
59	27.035277	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
62	28.042741	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
65	29.048631	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
68	30.049503	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
70	31.051683	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
71	32.066513	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
72	33.080848	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
73	34.086740	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
76	35.092689	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
78	36.094184	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]
79	37.107624	SpeedDra_b0:10:5d	SpeedDra_b0:10:d7	LLC	60	[Malformed Packet]


```

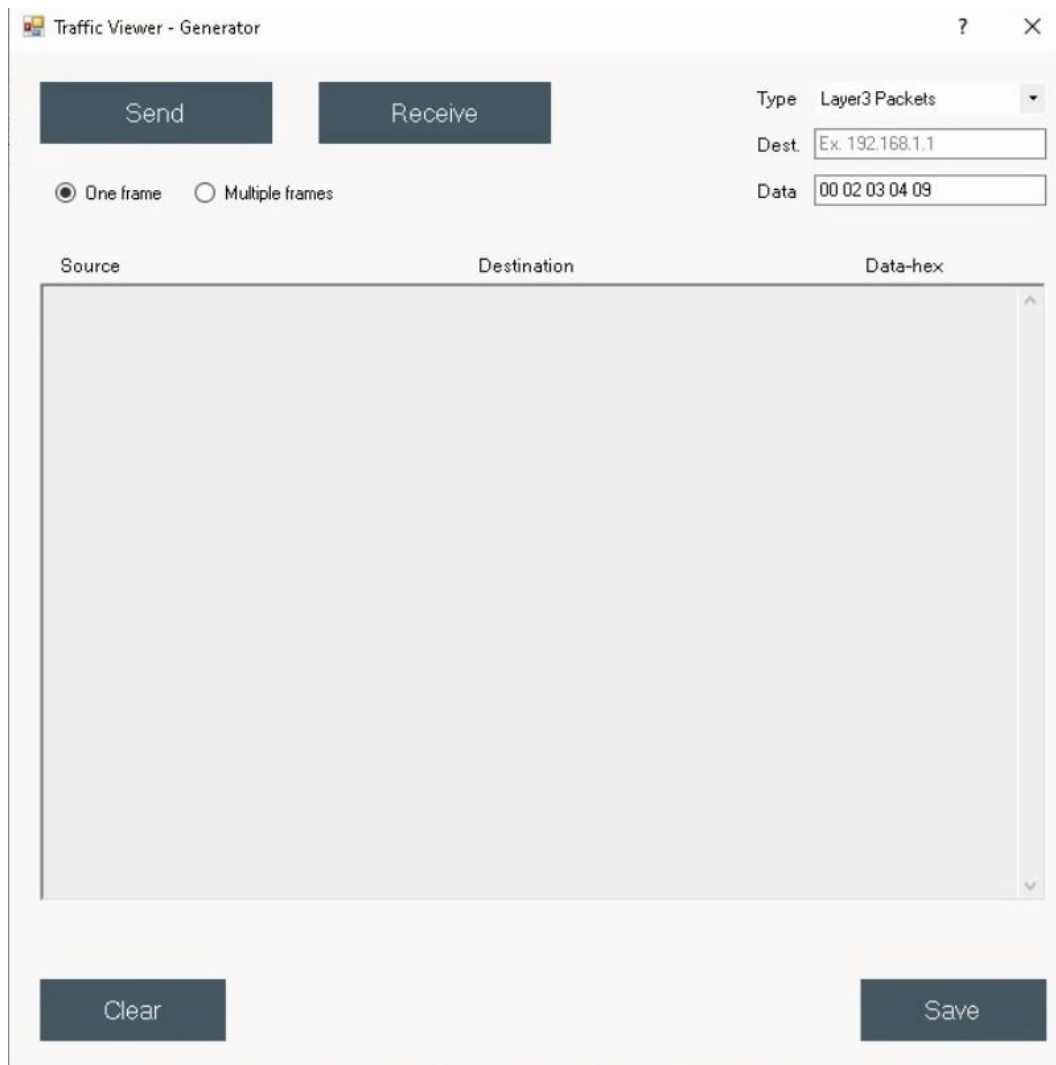
> Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF_{84824827-AADE-4BD2-82B8-8509}
> Ethernet II, Src: SpeedDra_b0:10:5d (00:13:b0:10:5d), Dst: SpeedDra_b0:10:d7 (00:13:b0:10:d7)
v 802.1Q Virtual LAN, PRI: 0, DEI: 49
    000. .... = Priority: Best Effort (default) (0)
    ...0 .... = DEI: Ineligible
    .... 0000 0011 0001 = ID: 49
    Type: 802.1Q Virtual LAN (0x8100)
v 802.1Q Virtual LAN, PRI: 1, DEI: 0, ID: 49
    001. .... = Priority: Background (1)
0000 00 13 b0 10 d7 00 13 b0 10 5d 91 00 00 31 ... ..]...1
0010 81 00 20 31 00 00 17 02 03 04 09 00 00 00 00 ... 1.....
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ... ..
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ... ..

```

Part 4 - Project: ANDi Graphical User Interface

Introduction

We built a Graphical User Interface to send and receive layer2 frames and layer3 packets between two computers connected over a media gateway. The GUI was implemented using python programming language and Windows Forms which can be run on ANDi tool. This GUI also helps in visualizing the sent and the received frames and packets. The following is a screenshot of the GUI application (Traffic Viewer-Generator) :



The main features of this application are:

- 1- choosing between Layer 2 frames and Layer 3 packets to send or receive.
- 2- The user can enter the destination IP or mac address according to the type.
- 3- Data format check for destination IP or MAC addresses is also available.
- 4- Changing the data byte values of the payload.
- 5- sending one or multiple IP packets or Ethernet frames.
- 6- Sent and received packets and frames are displayed on the GUI as source, destination and the payload data in hexadecimal format
- 7- Saving the sent and received packets and frames in text files
- 8-clearing the packets on the GUI
- 9- the user can see the source IP and MAC addresses of the machine by clicking on the help button on the upper right corner.

GUI Controls:

Receive button: used whenever the user wants to receive Layer2 frames or Layer3 Packets.

Send button: used to send Layer2 frames or Layer3 packets depending on which type the user selects.

Clear button:

used to clear the sent or received packets or frames from the GUI richTextBox.

Help button:

used to show the IP and MAC addresses for the selected adapter on ANDi.

Save button:

used to save the sent or received packets or frames in the GUI richTextBox to a text file which can be used for later analysis.

Exit button:

used to exit the GUI, it also shows a pop up dialog to save changes..

Destination textbox: used to hold the destination IP or MAC address.

Payload data textBox: used to hold the payload data in the hexadecimal format.

Type Combobox: allows the user to choose between Layer3 packets and Layer2 frames from a drop down list which will be later used for deciding the type to send or receive.

One frame radio button : used for sending single packet/frame.

Multiple frames radio button : used for sending multiple packets/frames.

Labels: are used as Headers on the GUI to distinguish the data in richTextBox and the purpose of controls(textboxes).

RichTextBox : used to display the source and destination IP or MAC addresses of the sent or received packets/frames as well as the payload data.

Implementation

Traffic viewer and generator application is based on Monolithic architecture. It has one main class which inherits windows forms and generates the GUI. We called this class PacketAnalyser as shown in the following code snippet:

```
class PacketAnalyser(Winforms.Form):  
    """Main class for GUI APP"  
  
    """
```

In addition to that it also contains methods to handle sending or receiving packets/Frames, saving documents, thread management etc. The constructor is used to build up the main form and all the components for the GUI by calling the create functions for each component which we implemented. The following snippet show how the constructor first create the main Window (setting size, colors, text etc.:

```
def __init__(self):  
    """Constructor to initilize the GUI"""  
  
    #setting Caption to GUI  
    self.Text = "Traffic Viewer - Generator"  
    #setting GUI color  
    self.BackColor = draw.Color.FromArgb(250, 250, 250)  
    #setting GUI size  
    self.FormSize = draw.Size(700, 700)  
  
    # create buttons and labels font and colors variables  
    self.buttonsFont = draw.Font("Lato", System.Single(12))  
    self.buttonsForeColor = draw.Color.FromName("White")  
    self.buttonsBackColor = draw.Color.FromArgb(255, 68, 90, 100)  
    self.labelsFont = draw.Font("Lato", System.Single(9))  
  
    # self.components = System.ComponentModel.Container()  
  
    #setting the minimum and the maximum size of the GUI  
    self.MinimumSize = draw.Size(700, 700)  
    self.MaximumSize = draw.Size(700, 700)
```

After it creates the main window , It creates the other components by calling the following functions:

```

#calling functions in this class to create the controls and add them to the GUI
self.createSendButton()
self.createReceiveButton()
self.createSaveButton()
self.createSelectFrameTypeBox()
self.createDestinationMacIpTextbox()
self.createSourceLabel()
self.createDestinationLabel()
self.createPayloadLabel()
self.createPacketsTextBox()
self.createOnePacketRadioButton()
self.createMultiplePacketsRadioButton()
self.createPayloadDataTextBox()
self.createPayloadDataLabel()
self.createMacIpLabel()
self.createClearButton()
self.createTypeLabel()

```

Each of these functions will create a control by instantiating objects of windows forms classes and setting the required attributes such as texts, colors, positions etc. The following code snippet for creating the SEND button gives an overview on how these functions were implemented..

```

def createSendButton(self):
    """Function to create a send Button : used for sending IP or MAC packets"""

    # creating button object
    self.sendButton = Winforms.Button()
    # setting location of button
    self.sendButton.Location = draw.Point(20, 20)
    # setting button size
    self.sendButton.Size = draw.Size(150, 40)
    # adding flat style to send button
    self.sendButton.FlatStyle = Winforms.FlatStyle.Flat
    #setting button's border
    self.sendButton.FlatAppearance.BorderSize = 0
    # Add button text
    self.sendButton.Text = "Send"
    # setting button font
    self.sendButton.Font = self.buttonsFont
    # setting Fore color
    self.sendButton.ForeColor = self.buttonsForeColor
    # setting Back color
    self.sendButton.BackColor = self.buttonsBackColor
    # adding on click event ( sendButtonOnClick function is triggered )
    self.sendButton.Click += self.sendButtonOnClick
    # adding button to the form
    self.Controls.Add(self.sendButton)

```

Thread Management:

Threads are major components of our application, they are used to handle processes in the background in which they help the GUI to run smoothly without getting freezed. One main thread is used for the main class, and two more for send and receive functionalities. In this project, We used the Threading library for thread management. And the following screenshots show the functions for the main thread which handles the main class.

```
from System.Threading import ApartmentState, Thread, ThreadStart

def appThread():
    """This function create object of the GUI and run it as Windows Forms application"""

    #creates object of PacketAnalyser
    app = PacketAnalyser()
    #running the object of PacketAnalyser as a Windows forms application
    Winforms.Application.Run(app)
    #disposing resources of PacketAnalyser object
    app.Dispose()

def main():
    """main function creates thread for the Windows Forms application(GUI)"""

    #thread to be scheduled for execution.
    thread = Thread(ThreadStart(appThread))
    #sets the apartment state of a thread to single threaded apartment
    thread.SetApartmentState(ApartmentState.STA)
    #start the thread
    thread.Start()
    #join the thread, to ensure that a thread has been terminated
    thread.Join()
```

We have done the same for both receive and send functions, so that when the send or receive buttons are clicked a new thread will get created. Once we have threads created we have to terminate or join the threads on the exit or when the user clicks the button again. This is done using boolean flags **exiting**, **receive button flag**, **send_button_flag** in the while loops inside the thread functions as shown below.

```

while receive_button_flag and not self.exiting:

    g_ethernet_msg.on_message_received += on_eth_msg_received
    g_ethernet_msg.start_capture()
    sleep(1)
    g_ethernet_msg.stop_capture()
    g_ethernet_msg.on_message_received -= on_eth_msg_received

```

exiting variable holds a True value if the user clicks on the exit button of the GUI, and the receiveButtonFlag holds 0 or 1. The main purpose of these conditions is to terminate the while loop in which it means to terminate the receive or send threads and not keep it running the background when the user clicks on the receive button again(to turn it OFF) or clicks on the exit button.

In addition to that, the Dispose() which runs on exit function will free up all the resources allocated for the application, so the thread will no longer be active.

```

#set the exiting bool value to True, which will in turn kill the active thread.
self.exiting = True

```

Receive frames/packets:

Once the receive button is clicked, the function `receiveButtonOnClick` gets called and it first checks the selected frame type which the user wants to receive from the other machine and then it starts a thread which will be used to handle the receiving process independently without affecting the behaviour of the GUI and other controls. This thread will run one of the functions (`receiveMacFramesthread`, `receiveIpPacketsthread`) according to the selected frame type.

`receiveIpPacketsthread` and `receiveMacFramesthread` functions will mainly check if the destination IP or MAC addresses of the received message matches the ip or the MAC addresses (Ethernet 2) of the machine, and if

they're equal , it will display the incoming IP packets or Ethernet frames information such as (source IP/MAC address, Destination IP/MAC address and the payload data) in the richTextBox which we will explain later.

In these functions there is a while loop with a time sleep for 1 second which captures messages every one second and this loop terminates as soon as the button is clicked again (which means it changes its state to OFF) or the exit button is clicked.

Send frames/packets:

When the send button is clicked , the function `sendButtonOnClick` gets called which first checks the state of the button(ON or OFF) and changes its behaviour accordingly. If the button was already OFF, it will first check the type of the frame (IP or MAC) and check the format of the destination IP or MAC address. After that it starts a thread which will handle the process in the background. Two main thread functions are used in this part and they are : `sendIpPackets` and `sendMacFrames`. These thread functions are selected according to the selected type .

The functions `sendIpPackets` and `sendMacFrames` check first the radio button values(one frame or multiple frames). If “one frame “ is selected , it will just send one frame or packet, otherwise it will start the thread with the while loop. This loop has a time sleep for 2 seconds in which it sends a packet/frame after every 2 seconds and it adds the packet/frame information(source,destination and payload data) to the richTextBox. This thread is terminated in the same exact way as the receive button.

Moreover , other functionalities were implemented such as changing the colors for these buttons (send , Receive) to indicate the state of each button(ON , OFF) ,changing their text and disabling other buttons when one of these buttons is ON.

Data and File management:

When the Save button is clicked, the function `saveButtonOnClick` gets called in which it calls another function for saving the document and it's called `SaveDocument()`. The following is a code snippet for `SaveDocument()` function:

```
def SaveDocument(self):
    """This function fetch the data on the richTextBox and save it as a file."""

    # Save document dialogue box to get the filename
    if self.saveFileDialog.ShowDialog() != Winforms.DialogResult.OK:
        return
    # data will be saved as "filename".txt
    filename = self.saveFileDialog.FileName
    # data will be saved as "filename".txt
    filename = filename.lower()
    # select the range of text in the richTextBox
    self.richTextBox.Select(0, 0)
    # Open the file in write mode
    stream = File.OpenWrite(filename)

    "My IP : {0} \nMY MAC: {1}", "Info".format(self.myIp, self.myMac)
    #set the data as header and richTextBox content.
    data = "{0:<29}{1:<25}{2}".format("Source", "Destination", "Payload") + "\r\n" + self.richTextBox.Text
    #set the data as byteStream of data
    data = System.Text.Encoding.ASCII.GetBytes(System.String(data))
    #write the file with byteStream content
    stream.Write(data, 0, data.Length)
    #close the file
    stream.Close()
```

The function opens the file in write mode and fetches the data from the richTextBox, along with appended headers, it will get written to the file and which will be saved with the specified file name.

Challenges

Server and software related :

ANDi: ANDI wasn't working at some times and was crashing multiple times during our workshop.

Python version: ANDi uses old version of python, which was causing many issues while doing the project

Network related: Network connection was slow , so we had to work locally to develop the GUI(even had to use Windows VM since we use Linux and pythonnet package was not working with Linux)

Common Workstation: while Some other students were working on the same workstation, we couldn't connect most of the times (worked locally on virtual machines)

Conceptual :

Understanding the vlan concept : We could learn about Tagging, Untagging, Media Gateway configuration for vlan mode.

Thread management : We were working with threads to run the GUI, receive or send threads parallelly. We had to develop flags to manage the thread life cycle.

Work distribution

<i>TASK</i>	<i>Majd Hafiri</i>	<i>Vipin Koshy Thomas</i>
Practical work parts (1,2,3)	shared	
Practical work part 4	Thread Management (receive)	Thread Management (send)
	GUI development (receive part)	GUI development (send part)
	Saving functionality (shared)	
	Brainstorming & GUI designing (shared)	
Documentation	<i>Part1</i> Layer2 Frames : send Layer3 packets: receive	<i>Part1</i> Layer 2 Frames: receive Layer 3 Packets: send
	<i>Part2:</i> receive	<i>Part 2:</i> send
	Part 2: VLANs (shared)	
	Part 3: (shared)	
	Conclusion,Challenges and References (shared)	

Conclusion

In this course , We have learned a lot about technologies such as Windows forms, ANDi tool, Wireshark, Traffic viewer and traffic generator tools of ANDi, VLANs etc.. from which we expanded our knowledge on network connectivity. And we were able to complete the 4 parts of the practical work regardless of the challenges and issues we had. The Project helped us to strengthen our knowledge in GUI designing and development , as well as understanding backend functionalities such as threading, sending and receiving packets/frames in a better way. We still have scope for improvement in our project, such as including Nmap functionalities, where users can analyse the network and identify the connected devices and send packets/frames to those devices.

References:

- [MediaGateway Documentation](#)
- [ANDi Documentation](#)
- [VLANS](#)
- [Windows Forms documentation](#)
- [Stackoverflow](#)
- [Pythonnet](#)