Embedded Connectivity Course Report

Prof. Dr. Ing. Andreas Grzemba

Ing. Johann Bretzendorfer

Group 2 Workstation 4

Rashed Al-Lahaseh – 00821573

Clavijo Epia German - 00808082

TABLE OF CONTENTS

PART 1

STARTUP WITH ANDI-TOOL AND LOOPBACK

TASK 1: RUNNING FIRST PYTHON SCRIPT

Analyze the python script, you should understand what happens if it's started.

This script delivers a packet to the loopback adapter and receives the identical packet back from the loopback adapter. The ethernet adapter is often used to transfer data, whereas the recorder adapter is typically used to receive data. We don't have proper communication equipment since we don't have it. On a single loopback adapter, the synchronization and logging interfaces are setup. This is a Windows operating system-created virtual Ethernet adapter. The script functions exactly as it does in the example project. Because we haven't created genuine two-way communication, there's no need to alter "custom mac."

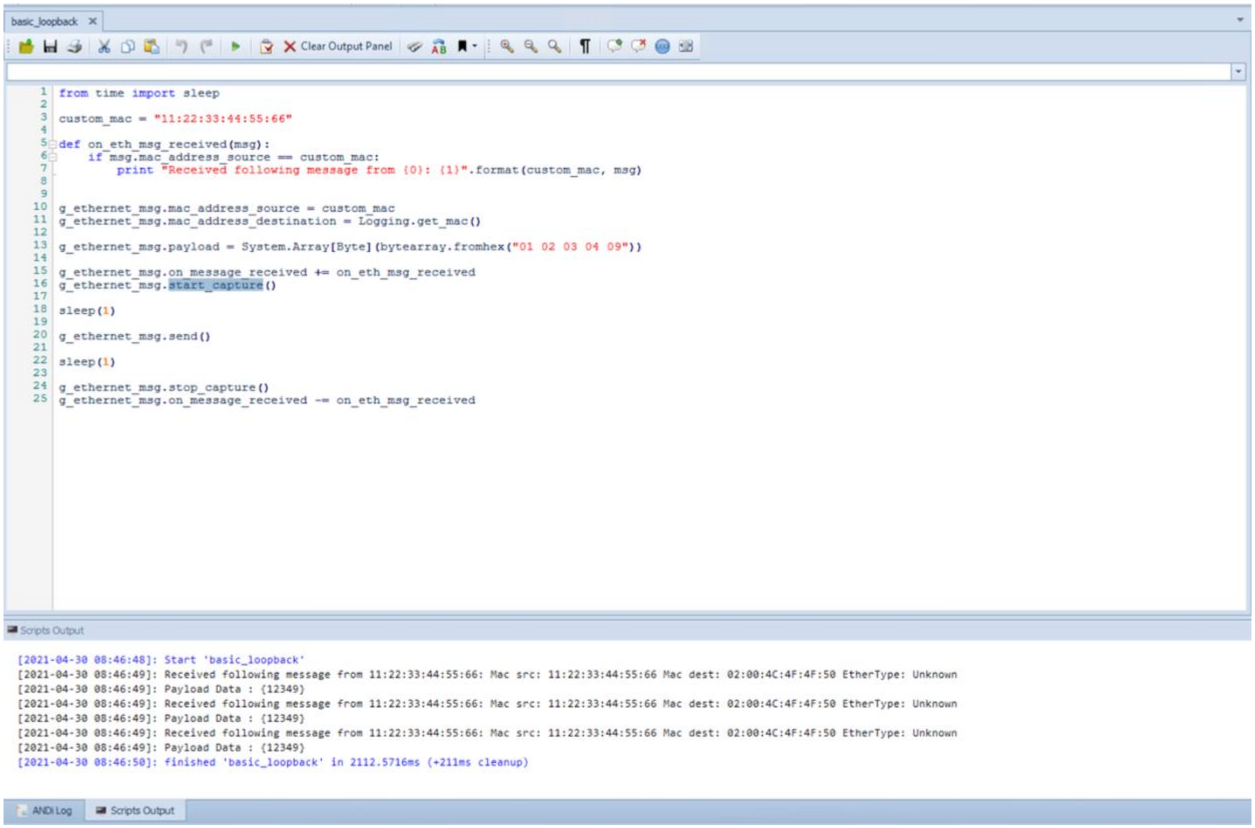


Figure 1: running first python script on Loopback adapter

TASK 2: FURTHER DIAGNOSTICS WITH WIRESHARK

For further diagnostic use Wireshark, try to set filter to get the frames that are interesting to you.

Wireshark is used to collect and analyze transmitted frames in the diagram below. We can see the source and destination MAC addresses, as well as the payload data “0102030409” that Loopback adapter has transmitted and received.
In addition, a source MAC filter is performed to show just the packages that are relevant to our investigation.

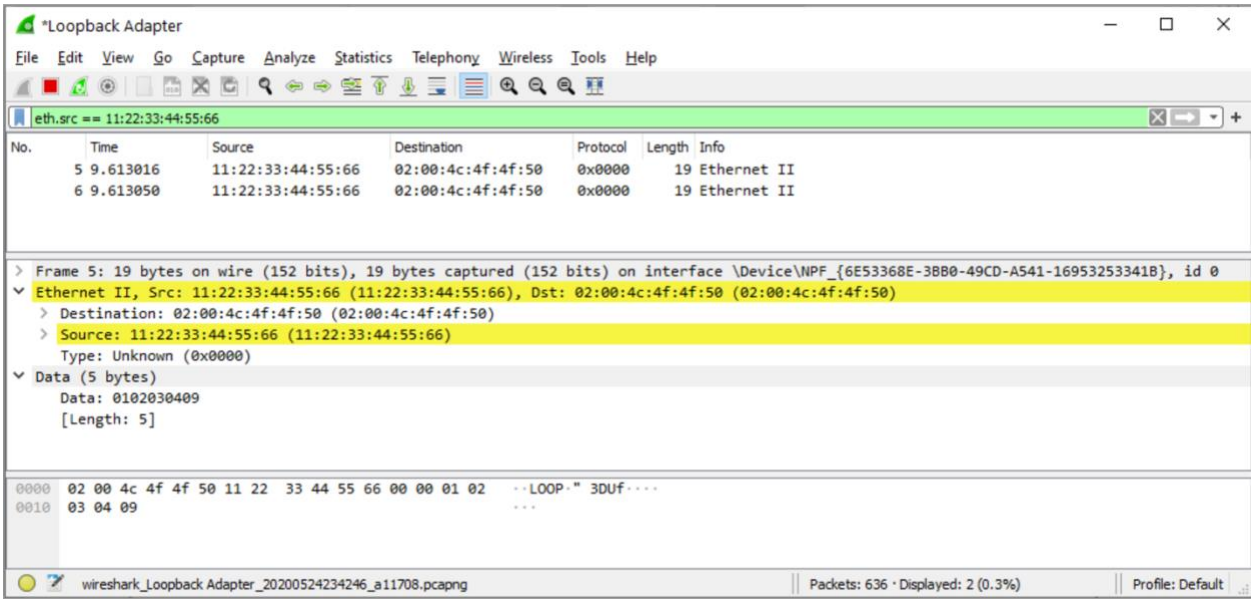


Figure 2: Analyzing frames sent by the first python script with Wireshark

TASK 3: SEPARATING SEND AND RECEIVE SCRIPTS

Based on the functionality of the basic-loopback write 2 new scripts - one to transmit a frame - one to receive a frame. (you have to start the scripts with the already known button, therefore it will be necessary to change some sleeptime-values).

1: USING TIMER FUNCTION IN SEND SCRIPT

If this works, define a timer function within your transmit program which will send an ethernet frame every second for about 20 seconds, to allow better verification increment the first byte of the payload-data.

2: REFINNING RECEIVE SCRIPT TO DETECT ALL SENT FRAMES

The receive script should be able to detect all frames you send.

The previous tasks were answered under the same script.

The main points of the sending script are:

- The script uses the timer function to send frames at 1000 millisecond intervals.
- The first byte of each frame of user data is a serial number for better inspection.
- Finally, the script sends a predefined message to the recipient.
- To write clean scripts, use try / except / final blocks in the script.

The main points of the receiving script are:

- Identify all frames sent by the script "Send" and "End of Transmission". Based on this mechanism, the receiver will continue to listen and receive frames until it receives {00} as data from the payload. After receiving {00}, the script will stop waiting and exit automatically, losing all frames in all timing conditions.
- To write clean scripts, use try / except / final blocks in them.

send.py

```python
#include libraries
from time import sleep

#used to add a sequence number to frames
count = 0

#set source and destination MAC addresses
g_ethernet_msg.mac_address_source = Simulation.get_ip()
g_ethernet_msg.mac_address_destination = Logging.get_ip()

#timer function
def on_elapsed(source, event_args):
    #inc counter
    global count
    count += 1
    #make message using value of count as it's first byte and then send it
    g_ethernet_msg.payload = System.Array[Byte](bytearray.fromhex('%02X'%count + "01 02 03"))
    g_ethernet_msg.send()
    #display a status message
    print("Sender: Frame" + '%02d'%count + " has been sent.")

try:
    #timer setup
    timer = andi.create_timer()
    timer.interval = 1000
    timer.on_time_elapsed += on_elapsed
    timer.start()
    #wait 20 seconds
    sleep(20)

finally:
    #send 'end of transfer' message : {0} this message ends receiver script
    print("Sender: Sending ""End of Transfer"" message to receiver...")
    g_ethernet_msg.payload = System.Array[Byte](bytearray.fromhex("00"))
    g_ethernet_msg.send()
    #different event
    timer.on_time_elapsed -= on_elapsed
    timer.stop()
```

Figure 3: Send Script

receive.py

```
#include libraries
from time import sleep

#this flag is used to exit the program when "end of transfer" message is detected: {0}
exitFlag = 0

#set source and destination MAC addresses
senderIP = g_ethernet_msg.mac_address_source = Simulation.get_ip()
g_ethernet_msg.mac_address_destination = Logging.get_ip()

#receving function
def on_eth_msg_received(msg):
    global exitFlag
    if msg.ip_header.ip_address_destination == senderIP:
        print "Receiver: Received message: [{1}]".format(senderIP, msg)
        if msg.payload[0]==0:
            exitFlag = 1

try:
    #set receiving function and start capturing
    g_ethernet_msg.on_message_received += on_eth_msg_received
    g_ethernet_msg.start_capture()
    while exitFlag == 0:
        sleep(1)
    #show an status message
    print("Receive: All frames received successfully. Ending receiver script ...")

finally:
    g_ethernet_msg.on_message_received -= on_eth_msg_received
    g_ethernet_msg.stop_capture()
```

Figure 4: Receive Script

Output:

```
[2021-04-30 11:45:21]: Start 'receive'
[2021-04-30 11:45:45]: Start 'send'
[2021-04-30 11:45:45]: Sender: Frame 01 has been sent.
[2021-04-30 11:45:45]: Receiver: Received message: [Mac src: 02:00:4C:4F:4F:50 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown [2021-04-30 11:45:45]: Payload Data : {1123}]
[2021-04-30 11:45:46]: Sender: Frame 02 has been sent.
[2021-04-30 11:45:46]: Receiver: Received message: [Mac src: 02:00:4C:4F:4F:50 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown [2021-04-30 11:45:46]: Payload Data : {2123}]
[2021-04-30 11:45:47]: Sender: Frame 03 has been sent.
[2021-04-30 11:45:47]: Receiver: Received message: [Mac src: 02:00:4C:4F:4F:50 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown [2021-04-30 11:45:47]: Payload Data : {3123}]
...
[2021-04-30 11:46:04]: Sender: Frame 20 has been sent.
[2021-04-30 11:46:04]: Receiver: Received message: [Mac src: 02:00:4C:4F:4F:50 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown [2021-04-30 11:46:04]: Payload Data : {20123}]
[2021-04-30 11:46:05]: Sender: Frame 21 has been sent.
[2021-04-30 11:46:05]: Sender: Sending End of Transfer message to receiver...
[2021-04-30 11:46:05]: Receiver: Received message: [Mac src: 02:00:4C:4F:4F:50 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown [2021-04-30 11:46:05]: Payload Data : {21123}]
[2021-04-30 11:46:05]: Receiver: Received message: [Mac src: 02:00:4C:4F:4F:50 Mac dest: 02:00:4C:4F:4F:50 EtherType: Unknown [2021-04-30 11:46:05]: Payload Data : {0}]
[2021-04-30 11:46:05]: finished 'send' in 20258.7164ms (+193ms cleanup)
[2021-04-30 11:46:06]: Receiver: All frames received successfully. Ending receiver script...
[2021-04-30 11:46:06]: finished 'receive' in 44937.036ms (+145ms cleanup)
```

TASK 4: ANDI TOOLS “TRAFFIC GENERATOR”, “TRAFFIC VIEWER”

Generate several Raw Ethernet frames with the ANDi ”Traffic Generator” and view it with ANDi tool ”Traffic Viewer”.

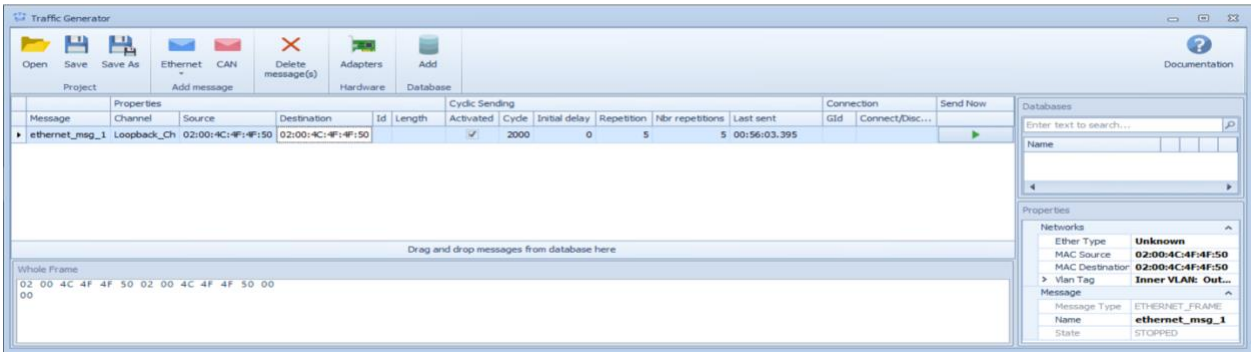


Figure 5: Traffic Generator tool, where it has produced and sent 5 raw ethernet frames within 2000 milliseconds interval

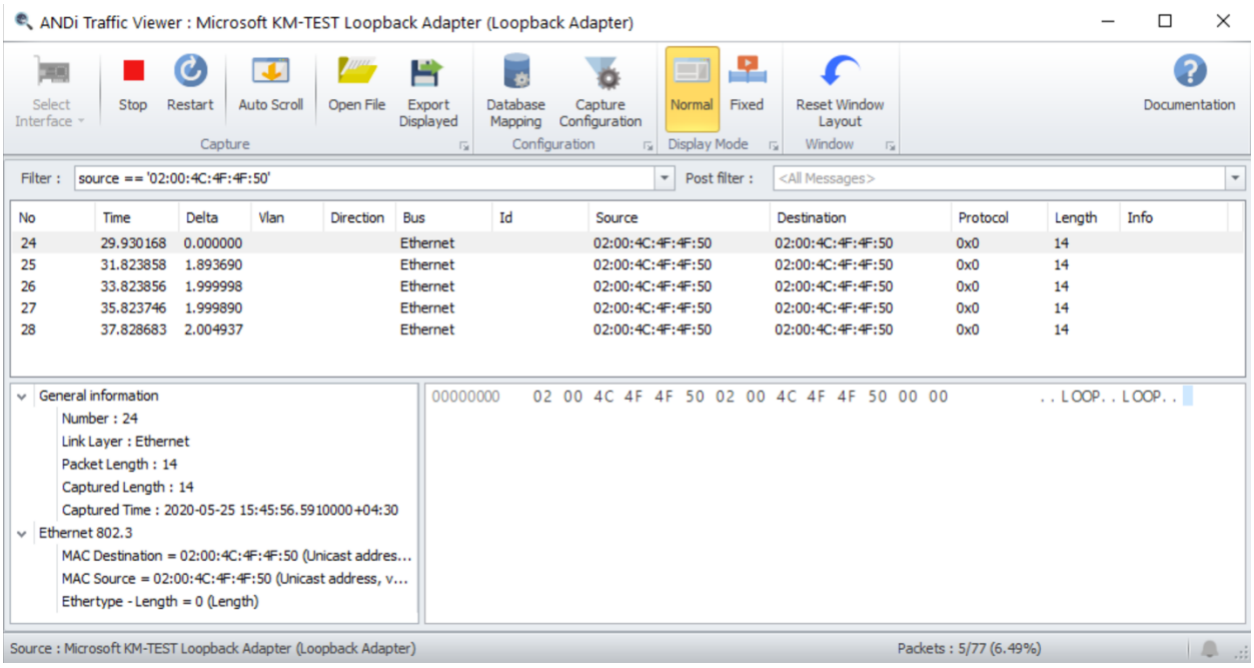


Figure 6: Traffic Viewer tool, which has received 5 frames send by the 'Traffic Generator' tool

TASK 5: ANDI TOOLS "SEND PING", "BURST SENDING" AND "IP/MAC"

Use the further ANDi tools like "Send Ping", "Burst Sending", "IP/Mac" ... and analyze the traffic.

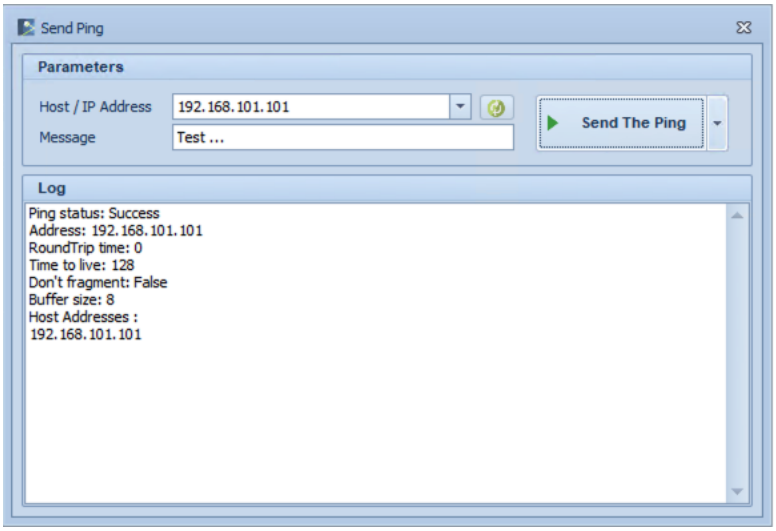


Figure 7: Send Ping tool, where it has been used here to ping PC1

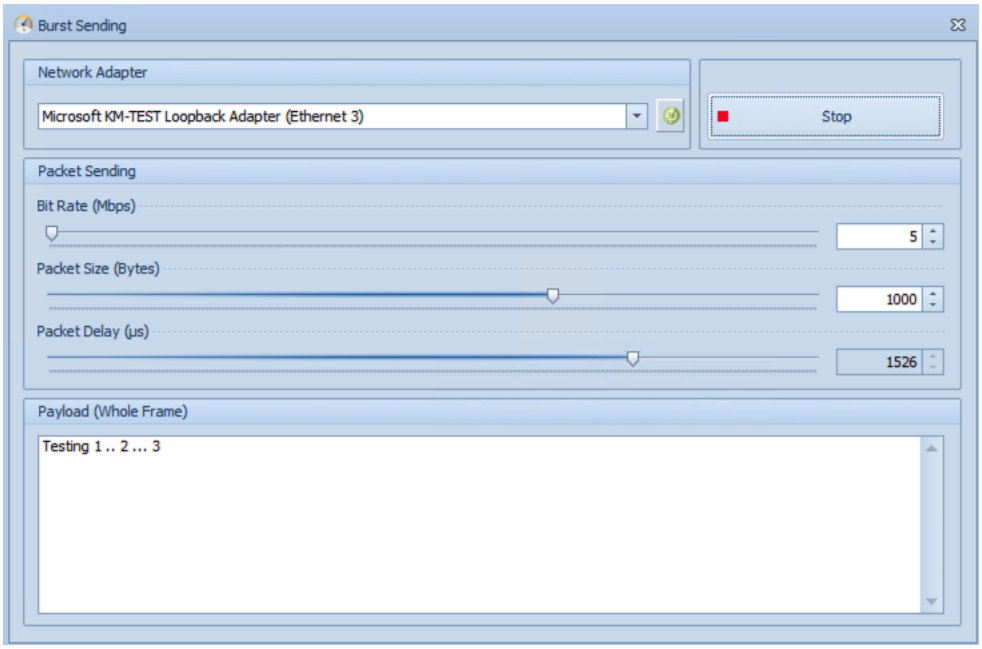


Figure 8: Burst Sending tool, which is used to send high amounts of packets to the network, for examble here it has been used to send 1000 Byte packets at 5 MBit/s

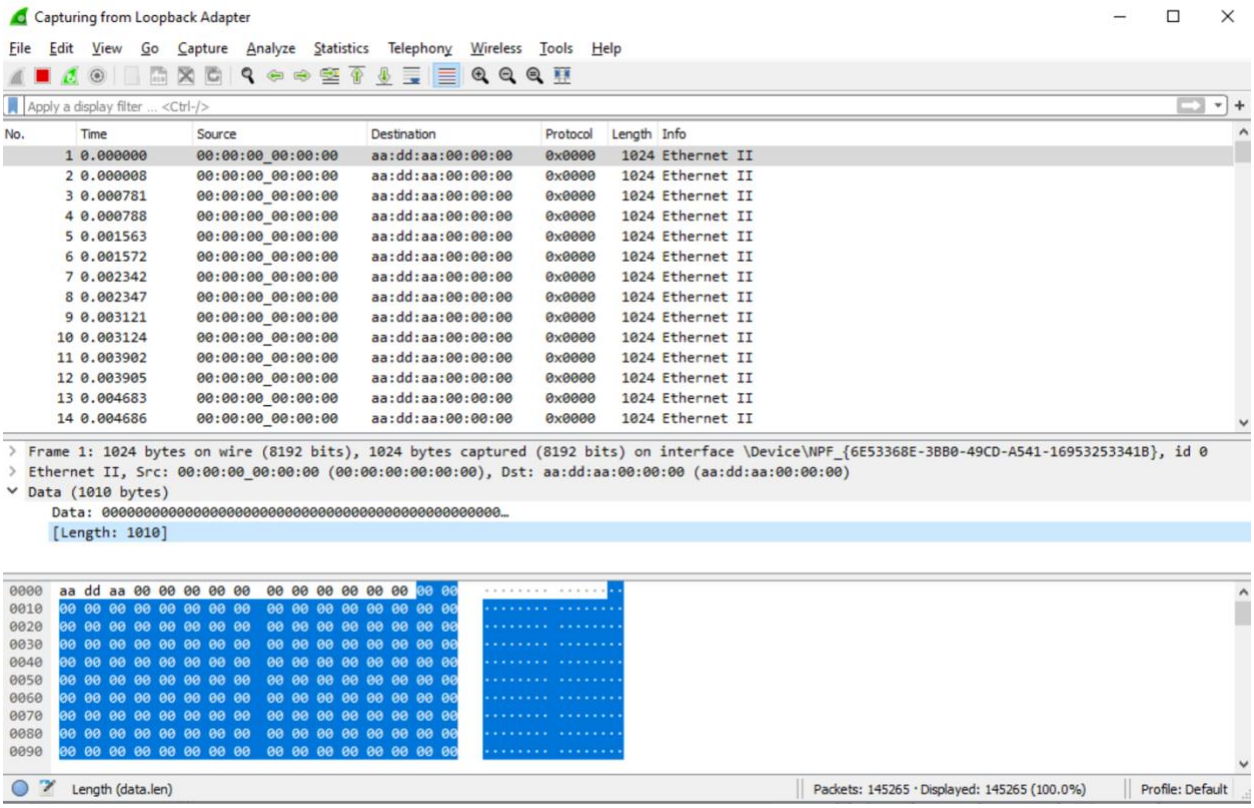


Figure 11: 'Burst Sending' traffic analyzed by Wireshark

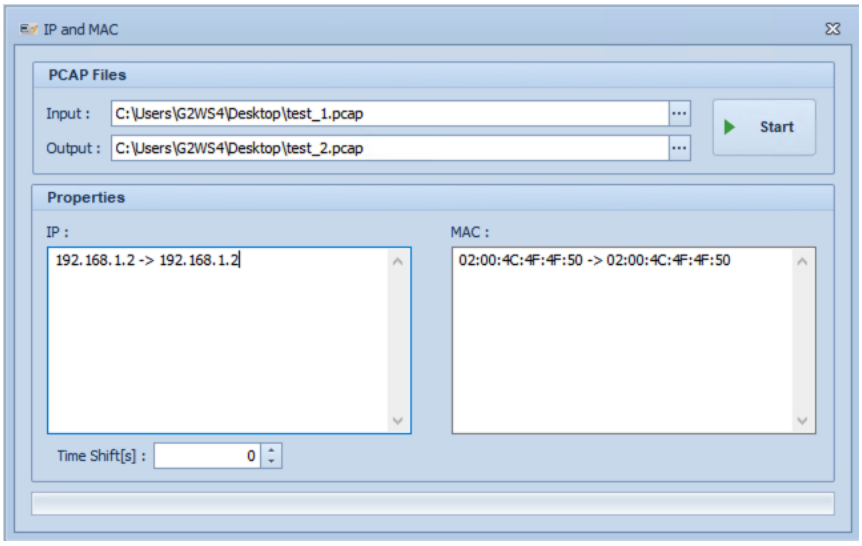


Figure 9: 'IP and Mac' tool, we can use 'Traffic Viewer' or 'Wireshark' to open the .pcap file. Then specify the output to be saved and the IP and Mac address of the message being transmitted. So we can have a new one. pcap", which contains the transmitted message, with the corresponding MAC address and source and destination IP addresses.

TASK 6: COMMUNICATION ESTABLISHING LAYER-3

In your scripts, change the communication from layer 2 (MAC address) to a layer 3 (IP address) connection. Analyze the data traffic with Wireshark. How do the frames change?

Maybe you have to setup the value for: "g_ipv4_msg_1.ethernet_header.mac_address_destination" with the Loopback MAC

In order to establish Layer 2 communication, so far we have used the MAC addresses of the source device and the destination device; our goal here is to send data packets to the device through the device's IP address; the specified communication is framed within the range of Layer 3-communication. The address "192.168.19.93" is assigned to the loopback adapter. The following script uses the g_ipv4_msg message for IP communication.

send.py

```python
#include libraries
from time import sleep

#used to add a sequence number to frames
count = 0

#set source and destination MAC addresses
g_ipv4_msg.ip_header.ip_address_source = Simulation.get_ip()
g_ipv4_msg.ip_header.ip_address_destination = Logging.get_ip()

#timer function
def on_elapsed(source, event_args):
    #inc counter
    global count
    count += 1
    #make message using value of count as it's first byte and then send it
    g_ipv4_msg.payload = System.Array[Byte](bytearray.fromhex('%02X'%count + "01 02 03"))
    g_ipv4_msg.send()
    #display a status message
    print("Sender: Frame" + '%02d'%count + " has been sent.")

try:
    #timer setup
    timer = andi.create_timer()
    timer.interval = 1000
    timer.on_time_elapsed += on_elapsed
    timer.start()
    #wait 20 seconds
    sleep(20)

finally:
    #send 'end of transfer' message : {0} this message ends receiver script
    print("Sender: Sending ""End of Transfer"" message to receiver...")
    g_ipv4_msg.payload = System.Array[Byte](bytearray.fromhex("00"))
    g_ipv4_msg.send()
    #different event
    timer.on_time_elapsed -= on_elapsed
    timer.stop()
```

Figure 10: Send Script

receive.py

```
#include libraries
from time import sleep

#this flag is used to exit the program when "end of transfer" message is detected: {0}
exitFlag = 0

#set source and destination MAC addresses
senderIP = g_ipv4_msg.ip_header.ip_address_source = Simulation.get_ip()
g_ipv4_msg.ip_header.ip_address_destination = Logging.get_ip()

#receving function
def on_eth_msg_received(msg):
    global exitFlag
    if msg.ip_header.ip_address_destination == senderIP:
        print "Receiver: Received message: [{1}]".format(senderIP, msg)
        if msg.payload[0]==0:
            exitFlag = 1

try:
    #set receiving function and start capturing
    g_ipv4_msg.on_message_received += on_eth_msg_received
    g_ipv4_msg.start_capture()
    while exitFlag == 0:
        sleep(1)
    #show an status message
    print("Receive: All frames received successfully. Ending receiver script ...")

finally:
    g_ipv4_msg.on_message_received -= on_eth_msg_received
    g_ipv4_msg.stop_capture()
```

Figure 11: Receive Script

Output:

```
[2021-05-01 14:33:15]: Start 'receive'
[2021-05-01 14:33:18]: Start 'send'
[2021-05-01 14:33:18]: Sender: Frame 01 has been sent.
[2021-05-01 14:33:18]: Receiver: Received message: [src: 192.168.19.93 dest: 192.168.19.93 Protocole: IP] [2021-05-01 14:33:19]: Sender: Frame 02 has been sent.
[2021-05-01 14:33:19]: Receiver: Received message: [src: 192.168.19.93 dest: 192.168.19.93 Protocole: IP] [2021-05-01 14:33:20]: Sender: Frame 03 has been sent.
[2021-05-01 14:33:20]: Receiver: Received message: [src: 192.168.19.93 dest: 192.168.19.93 Protocole: IP] ...
[2021-05-01 14:33:37]: Sender: Frame 20 has been sent.
[2021-05-01 14:33:37]: Receiver: Received message: [src: 192.168.19.93 dest: 192.168.19.93 Protocole: IP] [2021-05-01 14:33:38]: Sender: Frame 21 has been sent.
[2021-05-01 14:33:38]: Sender: Sending End of Transfer message to receiver...
[2021-05-01 14:33:38]: Receiver: Received message: [src: 192.168.19.93 dest: 192.168.19.93 Protocole: IP] [2021-05-01 14:33:38]: Receiver: Received message: [src: 192.168.19.93 dest: 192.168.19.93 Protocole: IP] [2021-05-01 14:33:38]: Receiver: All frames received successfully. Ending receiver script...
[2021-05-01 14:33:38]: finished 'send' in 20044.7159ms (+158ms cleanup)
[2021-05-01 14:33:39]: finished 'receive' in 24348.5679ms (+177ms cleanup)
```

Part 2

FIRST VIEW OF THE WORKPLACE

TASK 1: **SIMPLE CONNECTION OVER MEDIAGATEWAY**

In part 1 we have realized a simulated point to point connection with our loopback. Now we use the ANDi tool and the created scripts in a real network. In this step we want to establish a real point-to-point connection with the unconfigured media gateway as a Layer 2 switch.

STEP 1:

- CHECK WHETHER THE NETWORK CONNECTIONS (IP ADDRESSES) OF BOTH COMPUTERS ARE IN THE SAME CLASS C NETWORK (SUBNET).

- NOTICE THE MAC-ADDRESSES AND IPS OF BOTH NETWORK ADAPTERS (IPCONFIG /ALL), THIS INFO- MATION CAN ALSO BE FOUND IN THE APPENDIX - VERIFY!

So we run the following command to make sure of the addresses

```
C:Users\G2WS4: ipconfig /all
```

		PC1	PC2
Ethernet	Mac	80:89:A5:F2:BF:F3	80-89-A5-F2-BF-F1
	IP	192.168.101.101	192:168.101.115
Ethernet 1	Mac	00:13:3B:B0:0F:80	00:13:3B:B0:11:0A
	IP	192.168.0.209	192.168.0.210
Ethernet 2	Mac	00:13:3B:B0:0F:7F	00:13:3B:B0:11:09
	IP	192.168.0.229	192.168.0.230

STEP 2:

- START ANDI AND LOAD YOUR PROJECTS.
- UPDATE THE ANDI ADAPTER CONFIGURATION WITH THE APPROPRIATE NETWORK ADAPTERS, WE DON'T USE LOOPBACK ANYMOR
- "STIMULATION" ADAPTER IS FOR TRANSMITTING, "LOGGING" ADAPTER IS FOR RECEIVING.
- IN THE SCRIPTS FROM THE PREPARATION, YOU HAVE TO ADJUST THE MAC ADDRESSES (G_ETHERNET_ MSG.MAC_ ADDRESS_ DESTINATION AND G_ ETHERNET_ MSG.MAC_ ADDRESS_ SOURCE).

So bellow will find the updated versions of code for both scripts Sender/Receiver.

send.py

```
#include libraries
from time import sleep

#used to add a sequence number to frames
count = 0

#set source and destination MAC addresses
g_ethernet_msg.mac_address_source = "00-13-3B-B0-0F-7F"
g_ethernet_msg.mac_address_destination = "00-13-3B-B0-11-09"

#timer function
def on_elapsed(source, event_args):
    #inc counter
    global count
    count += 1
    #make message using value of count as it's first byte and then send it
    g_ethernet_msg.payload = System.Array[Byte](bytearray.fromhex('%02X'%count + "01 02 03"))
    g_ethernet_msg.send()
    #display a status message
    print("Sender: Frame" + '%02d'%count + " has been sent.")

try:
    #timer setup
    timer = andi.create_timer()
    timer.interval = 1000
    timer.on_time_elapsed += on_elapsed
    timer.start()
    #wait 20 seconds
    sleep(20)

finally:
    #send 'end of transfer' message : {0} this message ends receiver script
    print("Sender: Sending ""End of Transfer"" message to receiver...")
    g_ethernet_msg.payload = System.Array[Byte](bytearray.fromhex("00"))
    g_ethernet_msg.send()
    #different event
    timer.on_time_elapsed -= on_elapsed
    timer.stop()
```

Figure 12: Sender Script on PC1

receive.py

```python
#include libraries
from time import sleep

#this flag is used to exit the program when "end of transfer" message is detected: {0}
exitFlag = 0

#used to add a sequence number to frames
count = 0

#set source and destination MAC addresses
senderMAC = "00-13-3B-B0-0F-7F"
g_ethernet_msg.mac_address_destination = "00-13-3B-B0-11-09"

#receving function
def on_eth_msg_received(msg):
    global exitFlag
    if msg.mac_address_destination == senderMAC:
        print "Receiver: Received message: [{1}]".format(senderMAC, msg)
        if msg.payload[0]==0:
            exitFlag = 1

try:
    #set receiving function and start capturing
    g_ethernet_msg.on_message_received += on_eth_msg_received
    g_ethernet_msg.start_capture()
    while exitFlag == 0:
        sleep(1)
    #show an status message
    print("Receive: All frames received successfully. Ending receiver script ...")

finally:
    g_ethernet_msg.on_message_received -= on_eth_msg_received
    g_ethernet_msg.stop_capture()
```

Figure 13: Receiver Script on PC2

STEP 3:

SETUP VLAN - MEDIAGATEWAY

In order to use VLANs in MediaGateway, first, we need to activate VLAN feature:

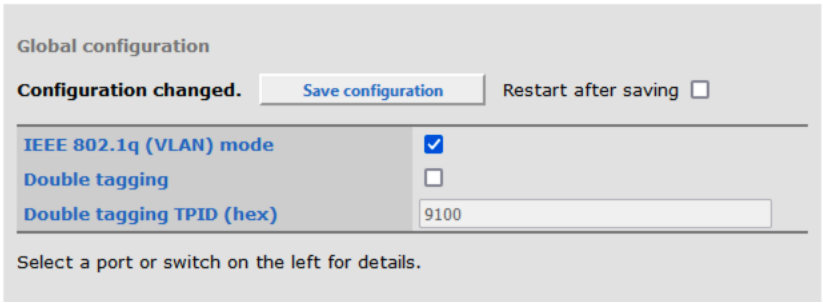


Figure 14: MediaGateway VLAN mode

S1-P1, which is connected to Ethernet 1 of computer 1, is used for MediaGateway config. S1-P4 is an internal port of MediaGateway. Both have same config as table below. We make them members of VLAN 049 with default VLAN ID 049 and as windows doesn't support VLAN, untag for both.

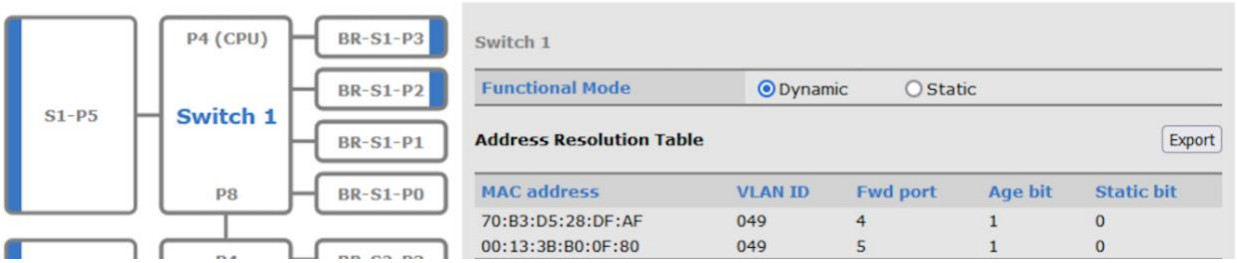


Figure 15: MediaGateway Switch 1 Configurations

PART 3

PCS CONNECTION THROUGH MEDIAGATEWAY

TASK 1: CONNECT THE TWO COMPUTERS VIA A VLAN BY PROGRAMMING THE MEDIA GATEWAY PORTS APPROPRIATELY AND TEST THE FUNCTIONALTY WITH YOUR TRANSMIT AND RECEIVE SCRIPTS

In order to establish connection between two computers VLAN 085 is defined on ports S1-P3 and S1- P2. The connection is made using Ethernet 2 of both computers.

Following figure shows status of switch 1 after applying the VLANs 049 and 085.

- VLAN 049 is defined to access configuration of MediaGateway.
- VLAN 085 is defined on Ethernet 2 of both computers to establish the connectivity between PC1 and PC2.

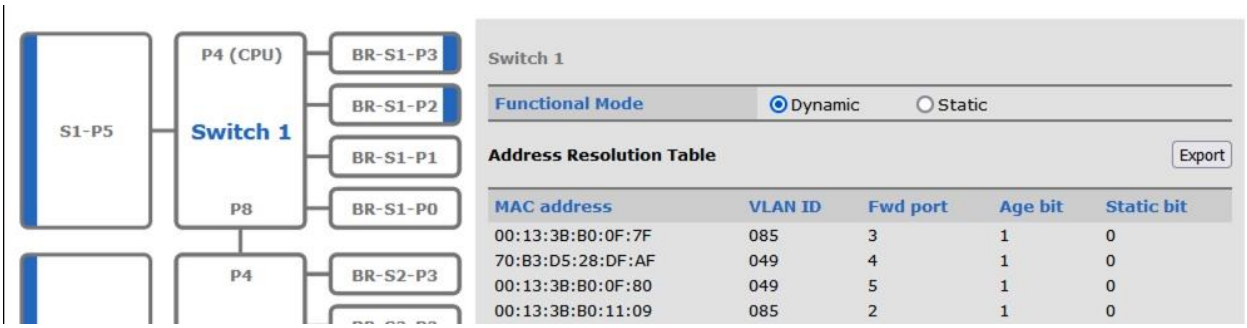


Figure 16: MediaGateway Switch 1 Configurations

The following figures show the receive logs after running the script again after MediaGateway setup.

Figure 17: ANDi tool logs

TASK 2: VERIFY THE TRAFFIC BETWEEN THE COMPUTERS VIA WIRESHARK THEN, CONNECT ONE OF THE TWO COMPUTERS TO THE WEBCAM

In order to gain access to the local webcam, ports S3-P8, S3-P4, S2-P8, S2-P4 and S1-P8 are configured to be a member of VLAN 85.

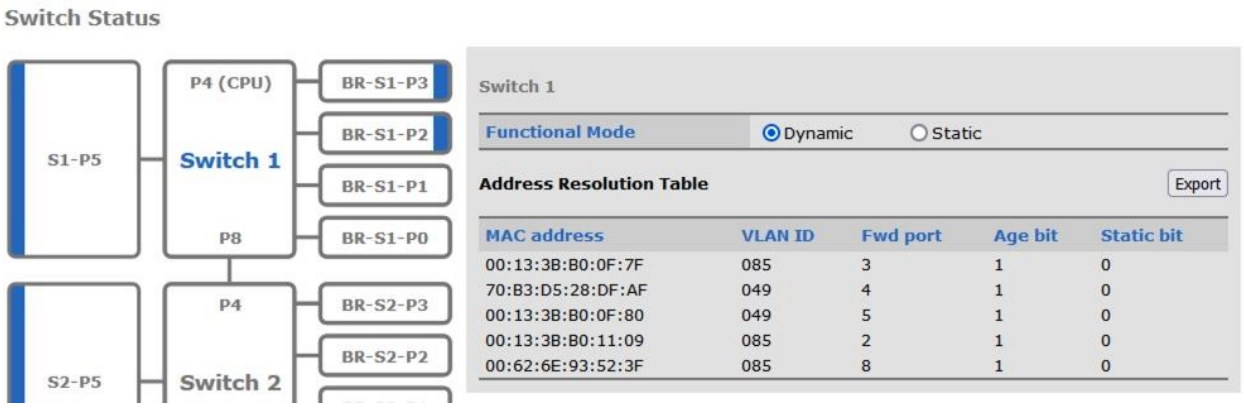


Figure 18: MediaGateway Switch 1 Configurations

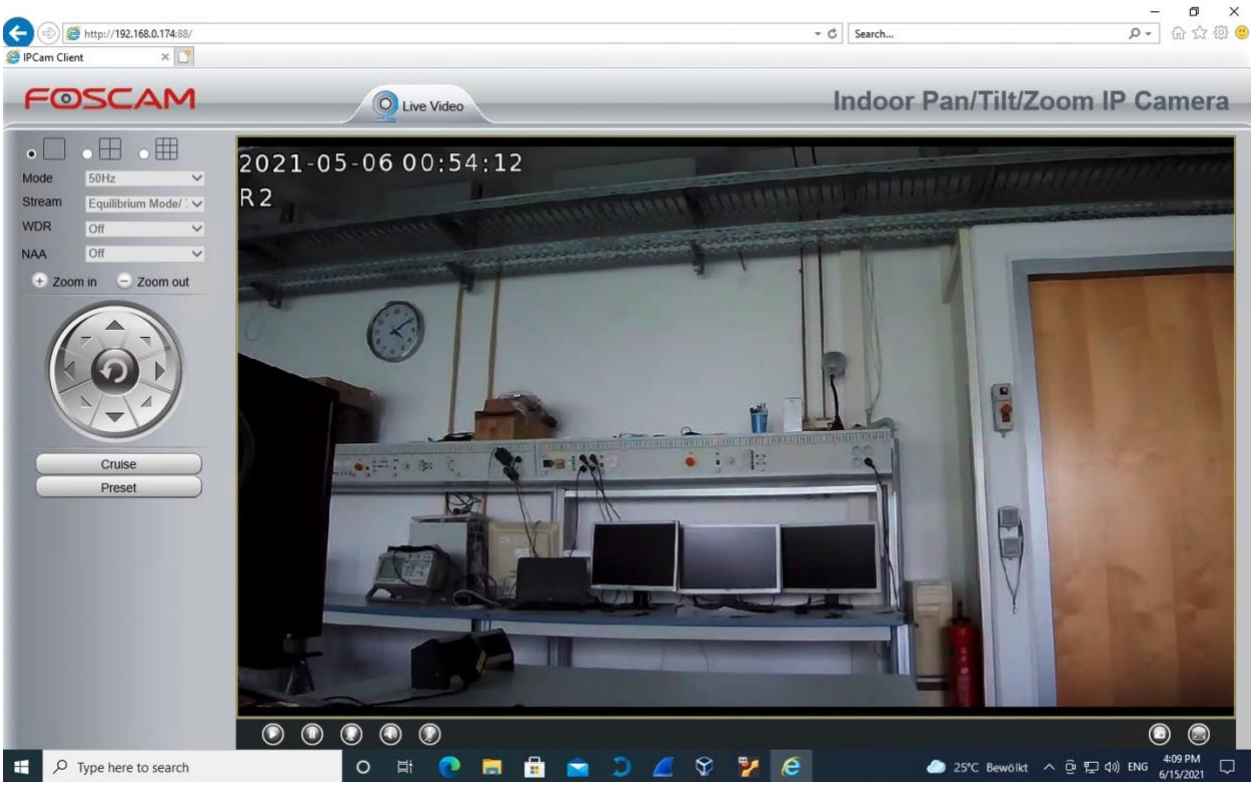


Figure 19: Local WebCam accessed through VLAN on PC2

TASK 3: ANDI TOOL AND AUTOMOTIVE ETHERNET IN THE LAB

Try to establish a connection across your MediaGatway and the central MediaGateway, for example, to display the pictures of central webcam. Then verify the traffic between the computers via Wireshark.

In order to fulfill the above-mentioned task, ports of media gateway have been configured in such a way that the PCs establish following connections:

1. PC1: Central Webcam – MediaGateway Configuration
2. PC2: Central Webcam – Local Webcam

Besides connecting to central webcam, we aim to separate local traffic from the traffic of central MediaGateway.

Following tables shows the corresponding configurations:

	S3-P8 (Local Webcam)	S3-P4 (Internal)	S2-P8 (Internal)	S2-P4 (Internal)	S1-P8 (Internal)
Default VLAN ID	070	--	--	--	--
VLAN Membership	070	070	070	073, 070	073, 070
VLANs to Untag	070	--	--	--	--

	S2-P3 (Central Gateway)	S1-P5 (PC1 – Ethernet 2)	S1-P4 (CPU)	S1-P3 (PC1 – Ethernet 2)	S1-P2 (PC2 – Ethernet 2)	S2-P5 (PC2 – Ethernet 1)
Default VLAN ID	073	049	049	073	070	073
VLAN Membership	073	049	049	073	070	073
VLANs to Untag	--	049	049	073	070	073

And the final step in order to establish physical connection with central MediaGateway, S2-P3 port needs to be set in 'Master' mode because the counterpart in central MediaGateway default value is 'Slave'.

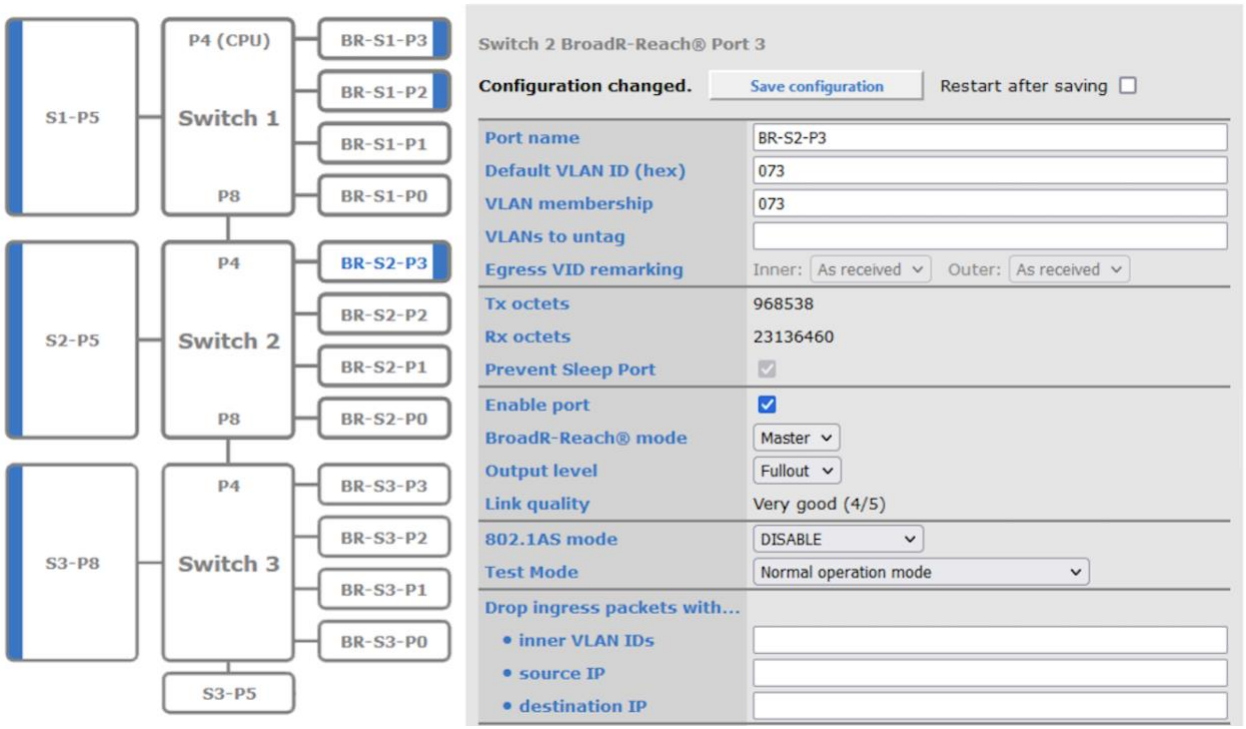


Figure 20: Port S2-P3 BoardR-Reach mode in Master mode

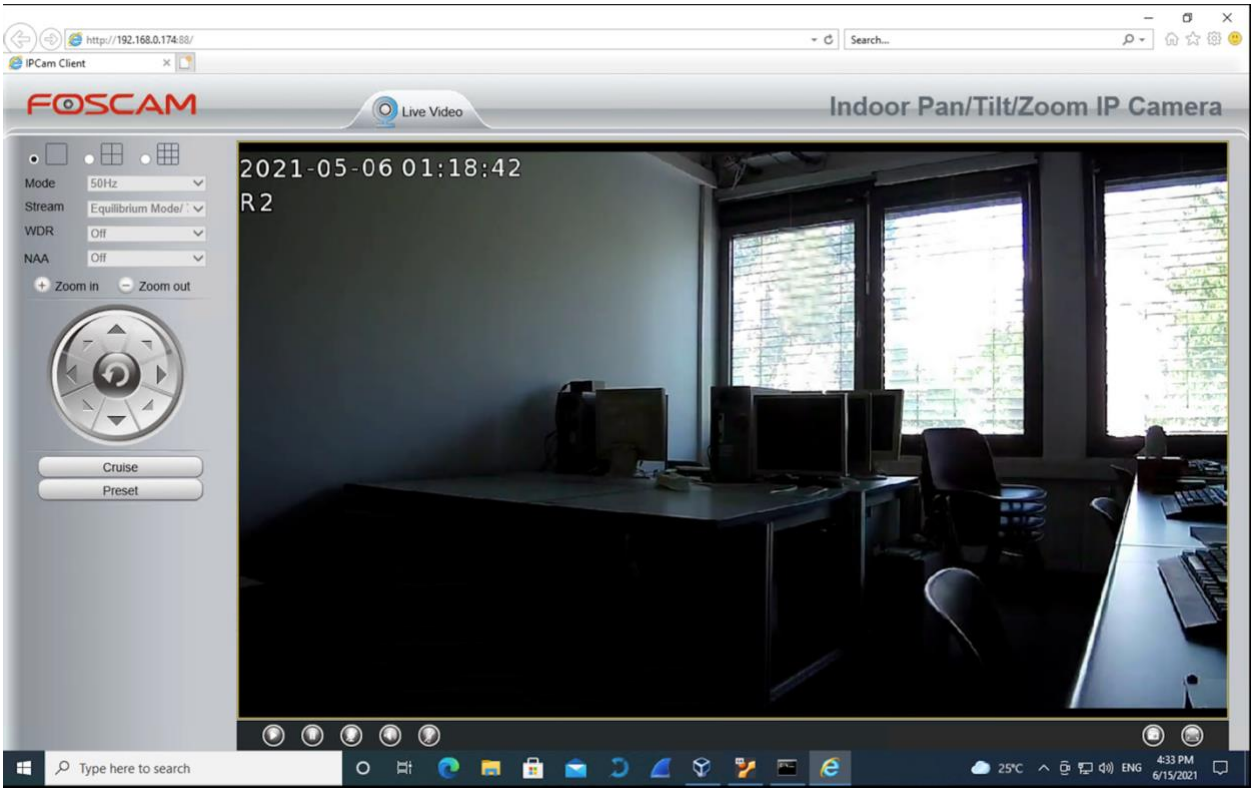


Figure 21: PC1 access to the Central Webcam

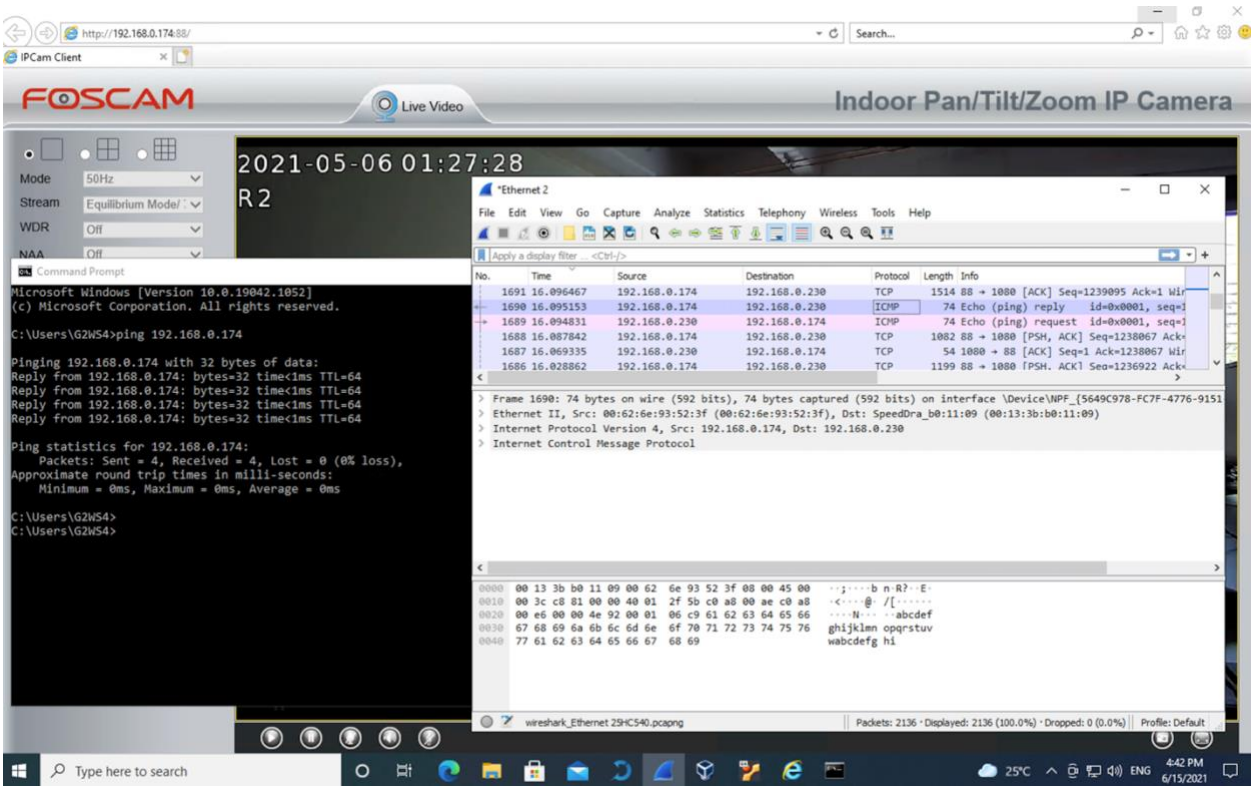


Figure 22: PC1 Wireshark analysis on Ethernet-2

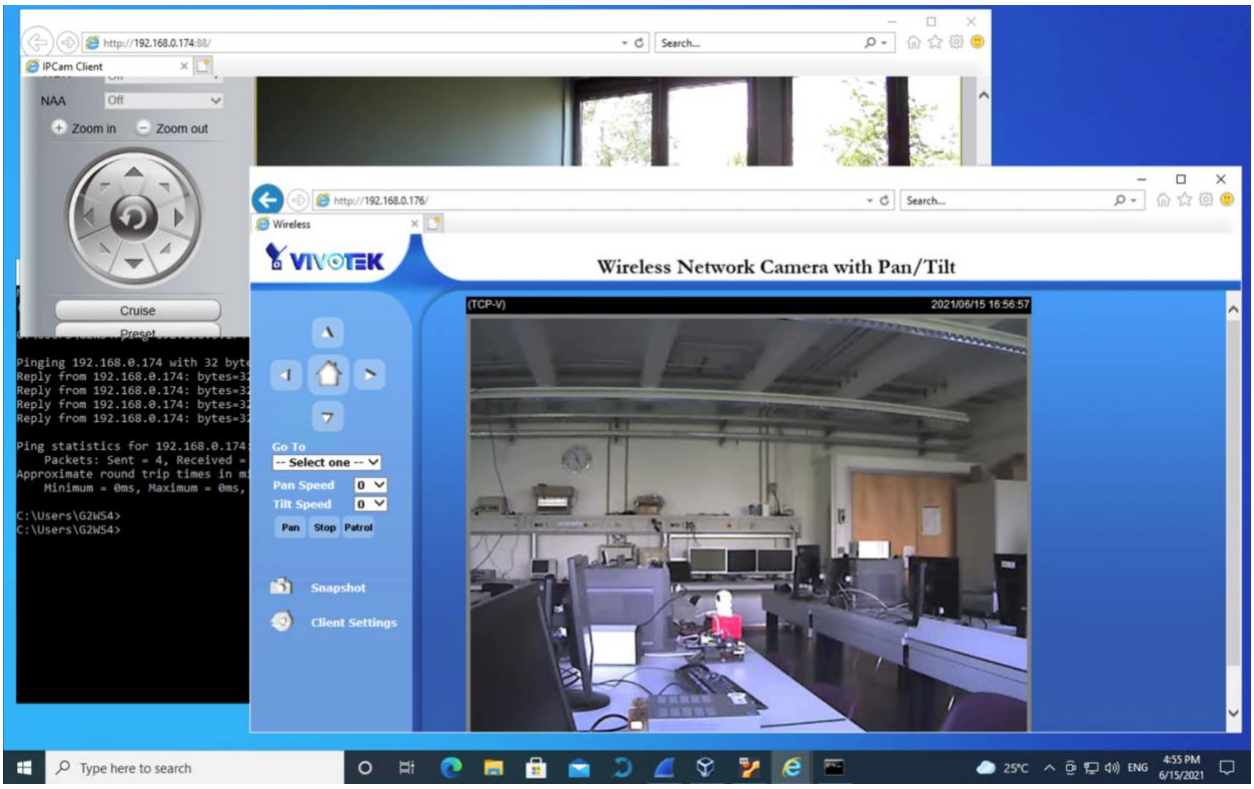


Figure 23: PC2 access to Central and Local Webcam

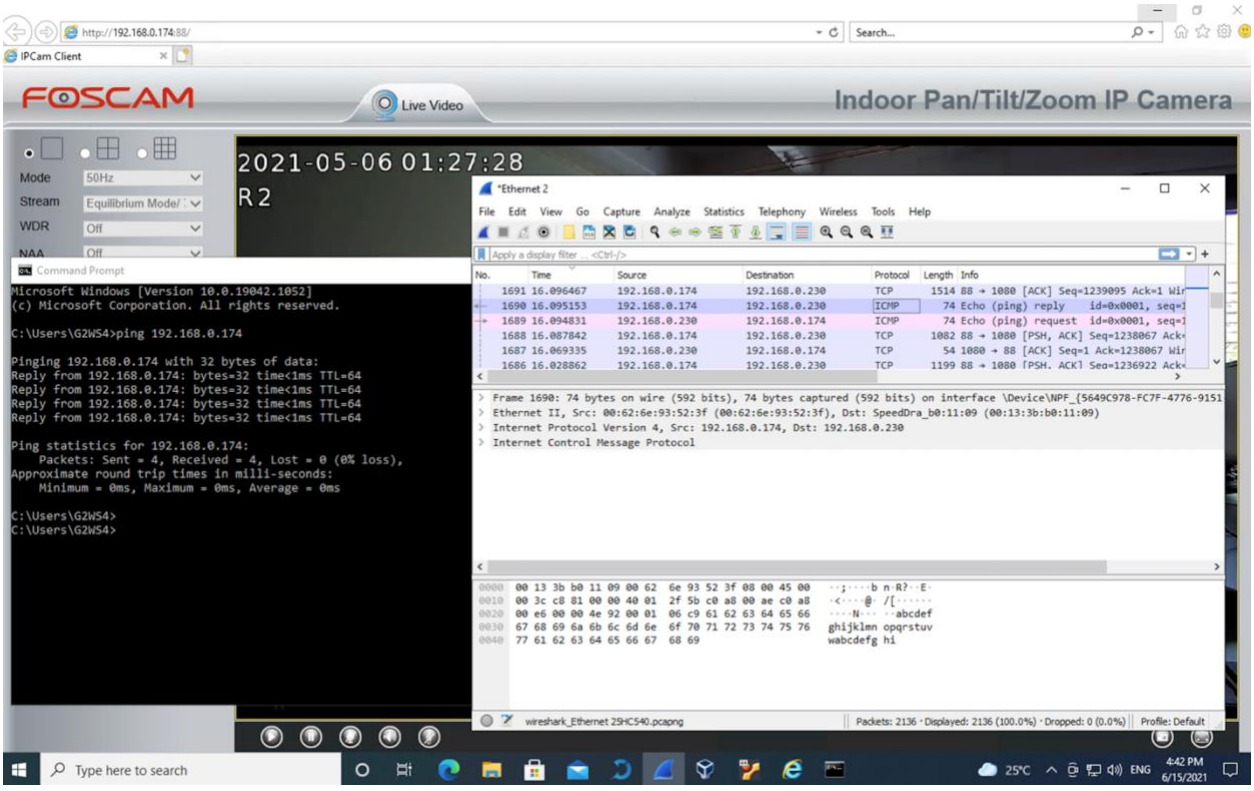


Figure 24: PC2 Wireshark analysis on Ethernet-2

And if we want to check the traffic captured by Wireshark between both PC1 and PC2 we will see that a MAC based communication is being performed. And the source/destination addresses demonstrates that the communication is taking place through a VLAN and based on our configuration VLAN-073 connects PC1-Ethernet 2 to PC2-Ethernet 1.

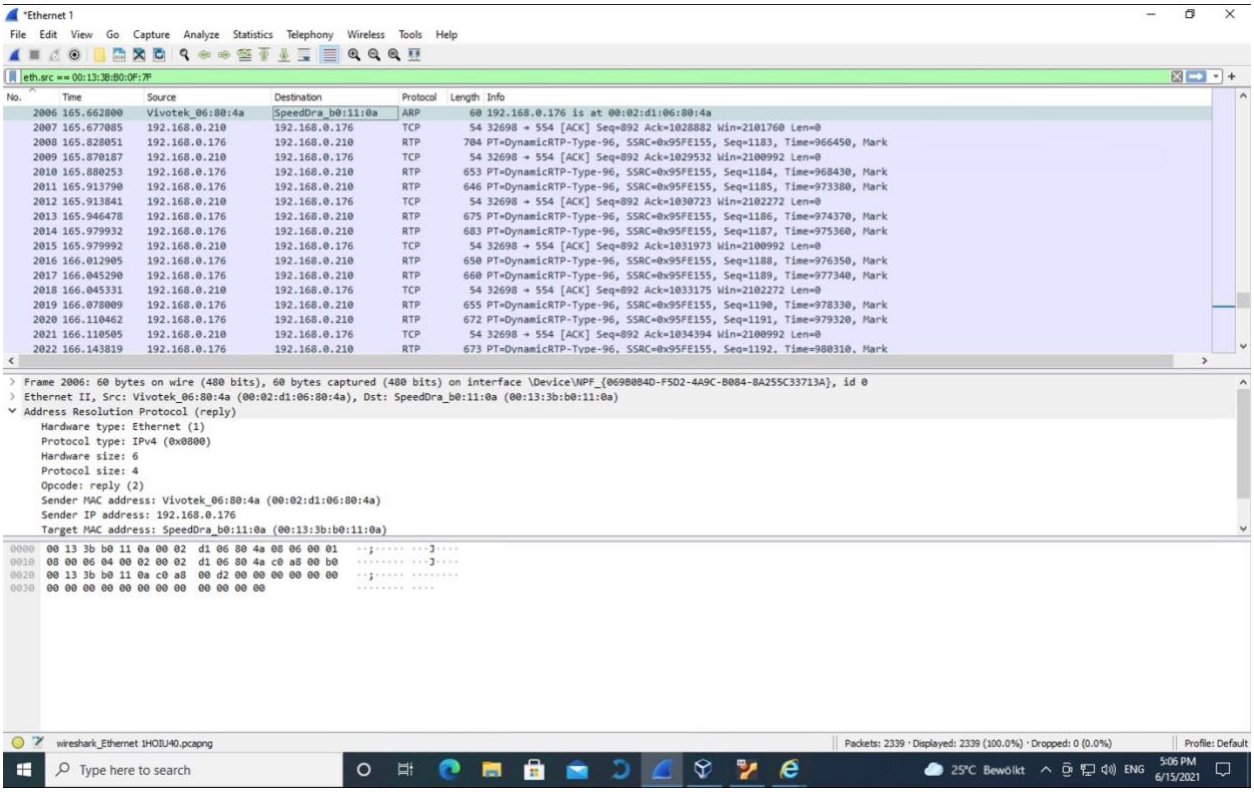


Figure 25: Wireshark analysis on PC2 - Ethernet 1

PART 4

VSOMEIP – FINAL PROJECT

VSomeIP
Possible on all workstations

Aim of the project:
A communication between 2 computers, connected by the media gateway, is to take place using the Someip protocol.

Setup:
The basis is the ”vsomeip” project of Bayerische Motoren Werke Aktiengesellschaft (BMW AG).
The vsomeip stack implements the http://some-ip.com/ (Scalable service-Oriented Midd- lewarE over IP (SOME/IP)) protocol. The stack consists out of:
• a shared library for SOME/IP (libvsomeip3.so)
• a second shared library for SOME/IP’s service discovery (libvsomeip3-sd.so) which is loaded during runtime if the service discovery is enabled.
The libraries are installed on 2 virtual LINUX computers. The two virtual computers (ser- ver/client) already communicate with a ”Hello World” example via a VirtualBox internal network connection. The description of vsomeip can be found at ”https://github.com/GENIVI/vsomeip/blob/master/documentation/vsomeipUserGuide” and further information about vsomeip at ”https://github.com/GENIVI/vsomeip/wiki/vsomeip-in-10-minutes”.

ToDo:
Your task is to install the virtual machines on 2 computers (user: labc109, pw:C109) and start the communication via the existing network (MediaGateway). The data exchange shall be logged with Wireshark and explained on the basis of the recor- ding.
In addition, the simulation of a node (server or client) can be done via ANDi.

STEP 1: COMMUNICATION BETWEEN 2 LINUX VMS

We will need at the begging to setup MediaGateway connection between PC1 and PC2 as we did in the past parts.

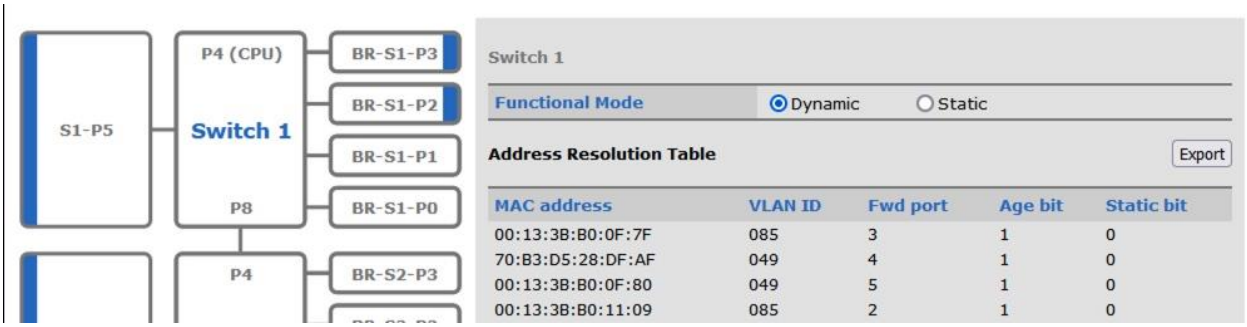


Figure 26: MediaGateway Switch 1 Configurations

Then we need to configure bridge connection for the VM adapter.

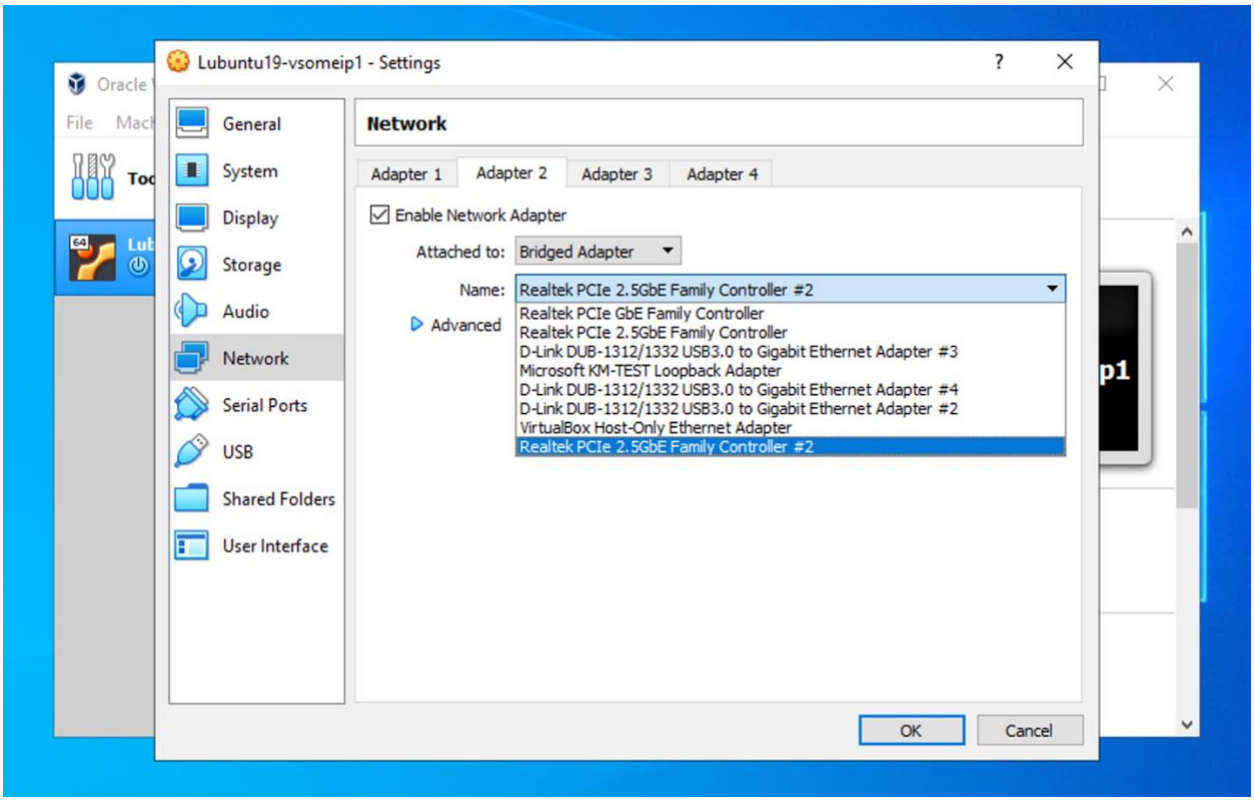
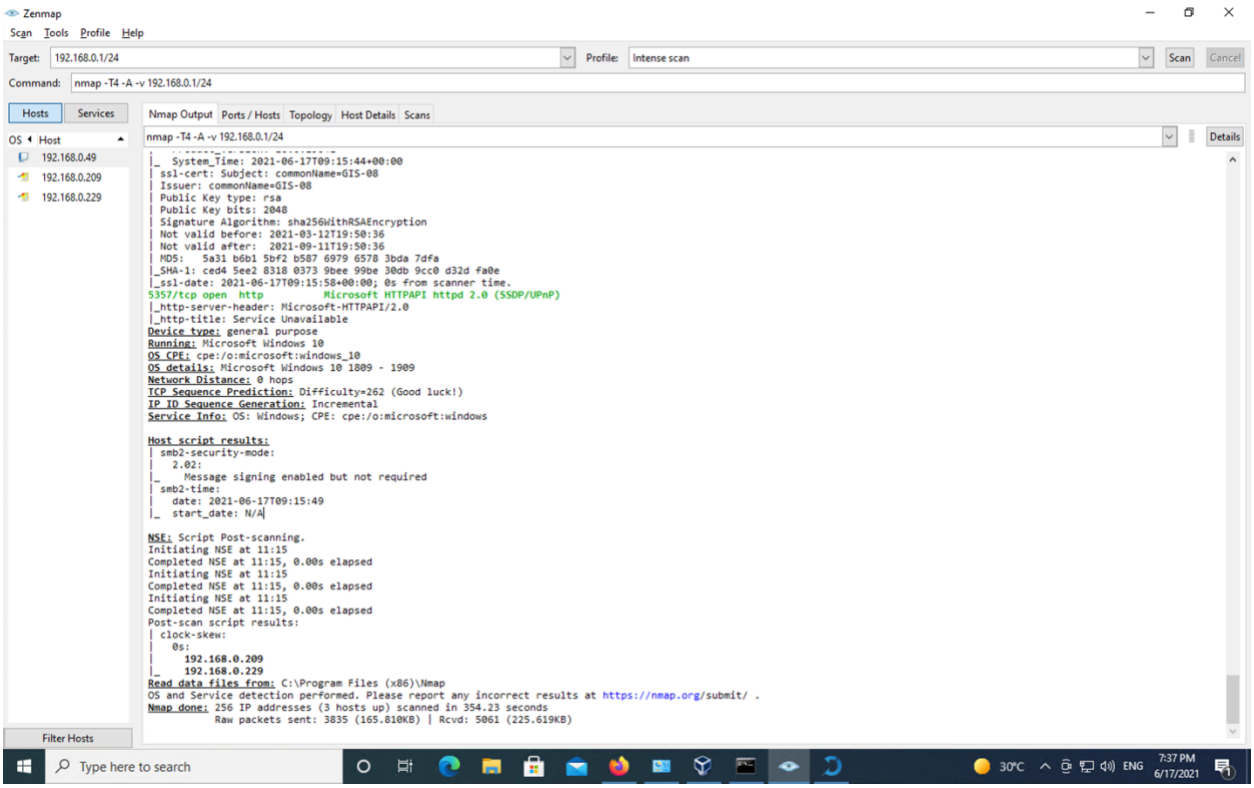


Figure 27: VM Adapter-2 Connection on PC1

Run nmap scan to make sure of the connections



Now by running the following command we can get the IP/MAC addresses for the machines

```
lab109@lab109-pc: ip a
```

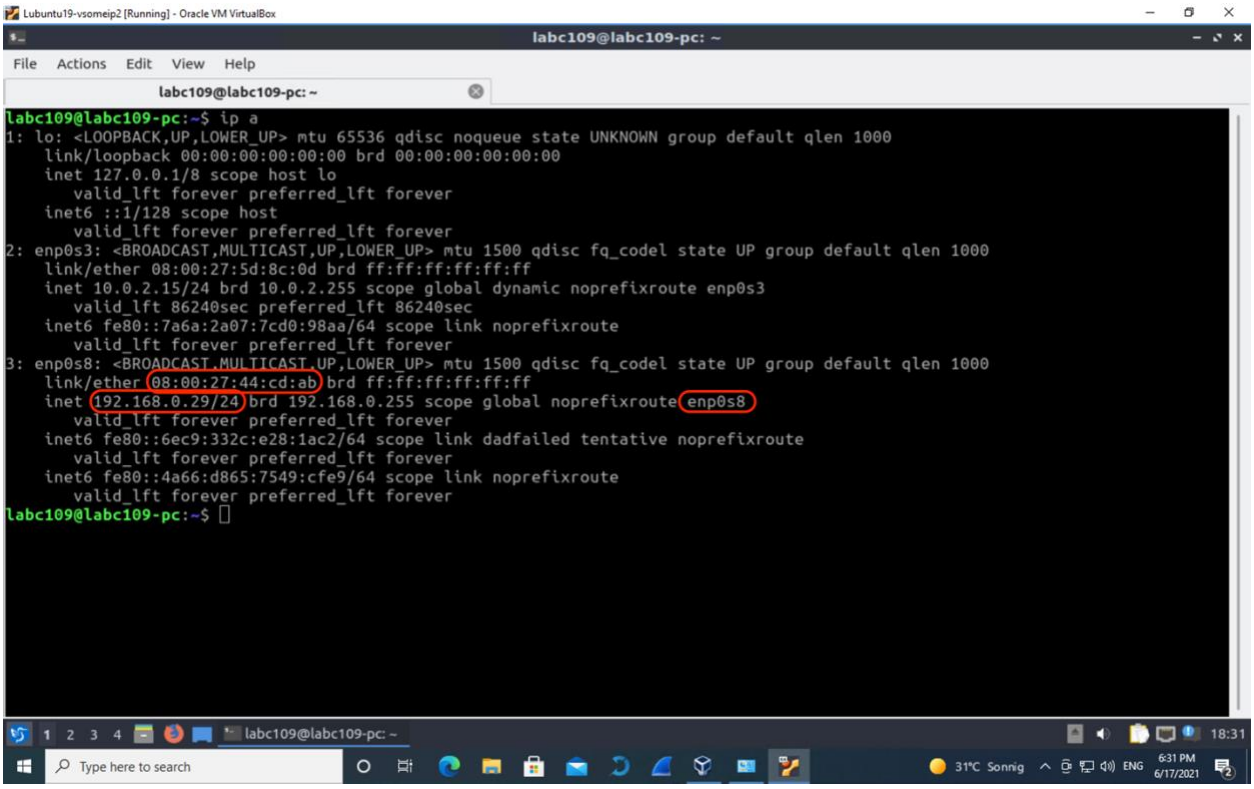


Figure 28: Network configurations on PC2 virtual machine

So, after retrieving the configurations we should have the following information

VM Name	IP	Mac Address
Lubunto19-vsomeip1 PC 1	192.168.0.28	08:00:27:6b:2a:9e
Lubunto19-vsomeip2 PC 2	192.168.0.29	08:00:27:44:cd:ab

STEP 2: INTALIZTING VSOMEIP

The 'VSomeIP' library were already installed on both VMs and inside it there are examples in which we can demonstrate communication between these devices.

Now for our task we are going to use the "hello_world" example.

Inside the example there are two C++ files for both Client and Service:

- hello_world_service.cpp, contains different methods for creating application, initialize, start, stop, terminate, offer service, create response and so on.
- hello_world_client.cpp, contains different methods for creating application, initialize it, start, request service, check availability, get the payload and print it and stop.

Also, other important files:

- CmakeLists.txt, creating executable files for the C++ files.
- helloworld-local.json, json format file used for configuration settings of the service/client applications.

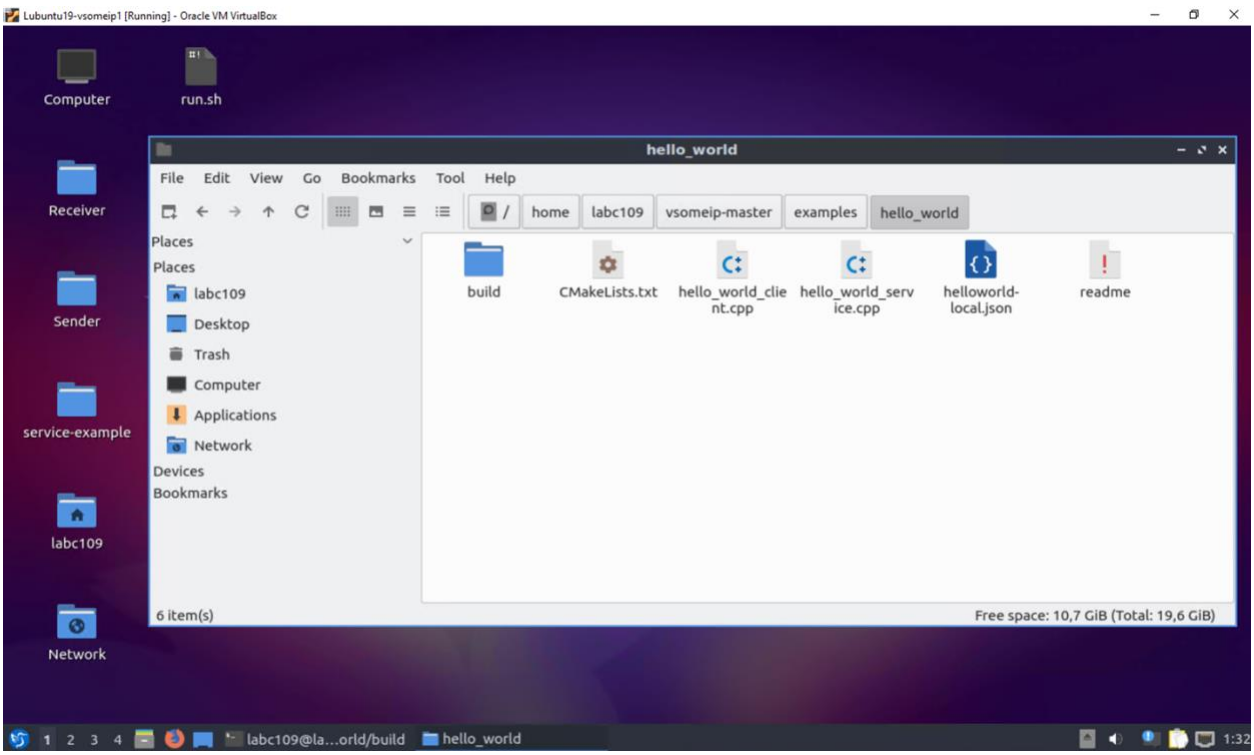


Figure 29: hello_world example folder

Now if we get back to the JSON configuration file we need to understand each filed and what does it refer to.

Service Side – PC1:

- Unicast, address should be set to PC1 IP address (192.168.0.28).
- Applications, the setting refer to the service application name which will be made by running the service file, so every application has a unique ID which called 'medthod_id' and here we chose it to be '0x3333' 16 bit.
- Services, refer to the setting specific for the service where the 'service' here is an id which is unique for each service and same for client nodes and for the 'unreliable' indicates that UDP protocol is used for under port '30509' where this service is available.
- Routing, this field is specific for each application as a routing manager.

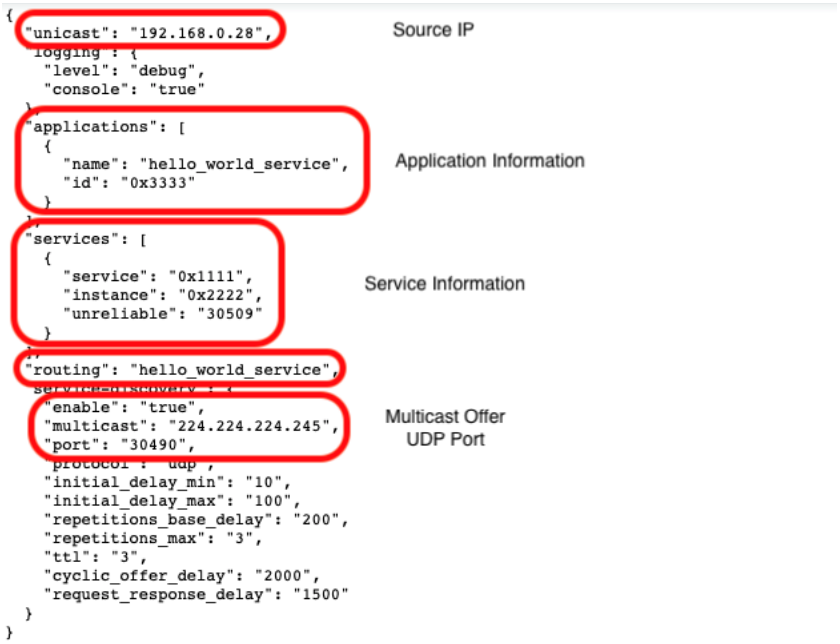


Figure 30: PC1 - JSON - Configurations

Client Side – PC 2:

The service section is not necessary here as there is only client node going to be running.

Figure 31: PC2 - JSON - Configurations

Then we run the following command to make sure that multicast messages connection is open for both PCs

```
lab109@lab109-pc: sudo route add -nv 224.224.224.245 dev enp0s8
```

And then we need to compile the example by build the C++ files and the applications by using

```
lab109@lab109-pc: cmake ..
```

```
lab109@lab109-pc: make
```

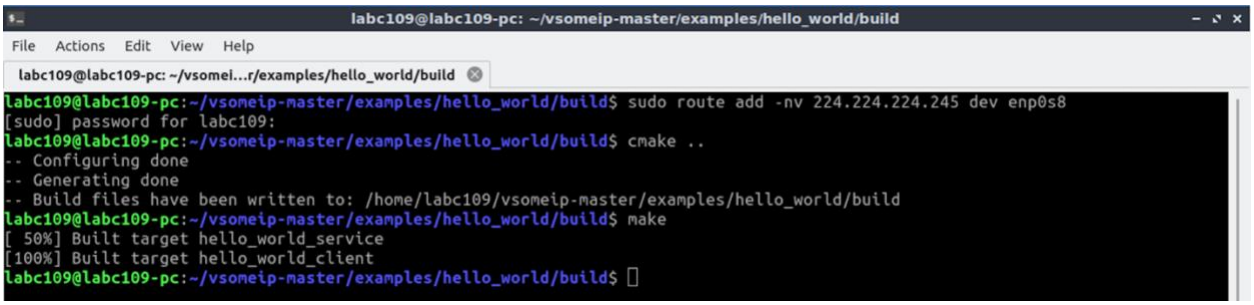


Figure 32: building 'hello_world' example

STEP 3: RUN APPLICATIONS

Now after we built the files in order to run the compiled examples, we need run the following commands.

PC1 – Service

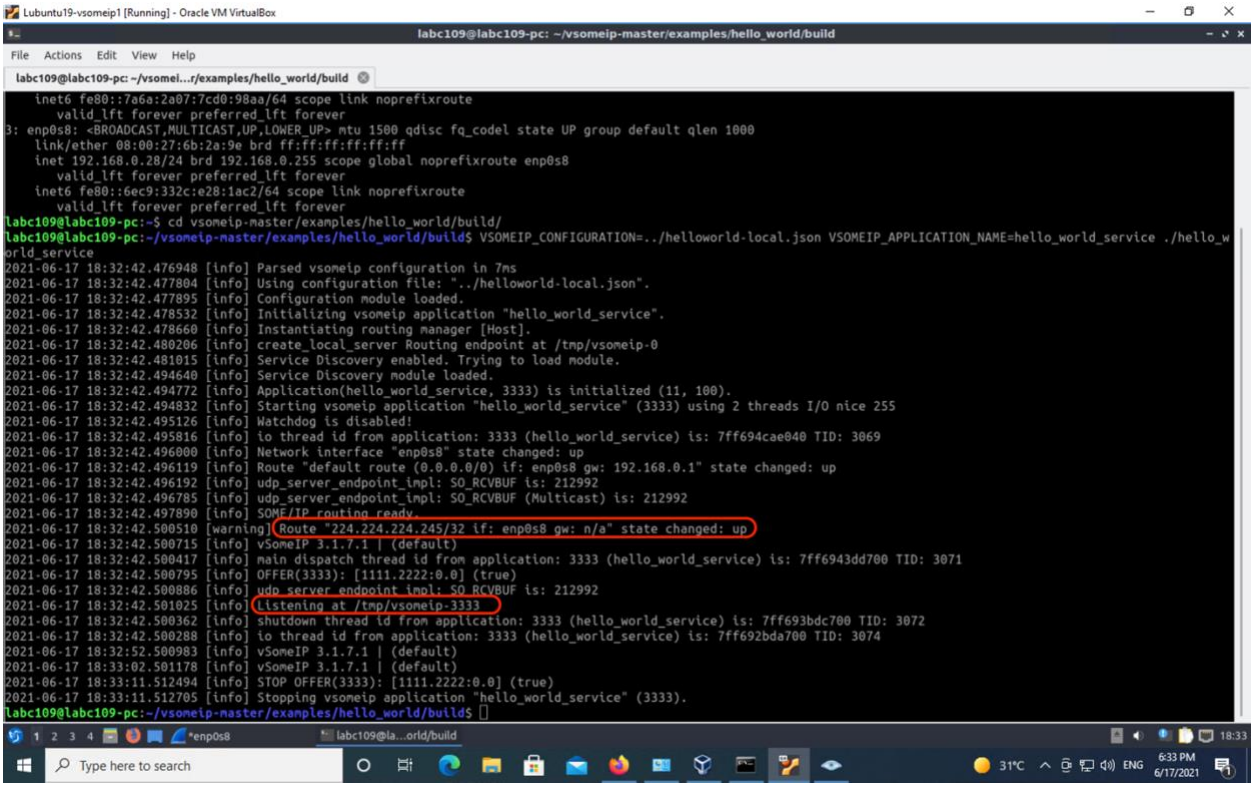
```
VSOMEIP_CONFIGURATION=../helloworld-local.json \
VSOMEIP_APPLICATION_NAME=hello_world_service \
./hello_world_service
```

PC2 – Client

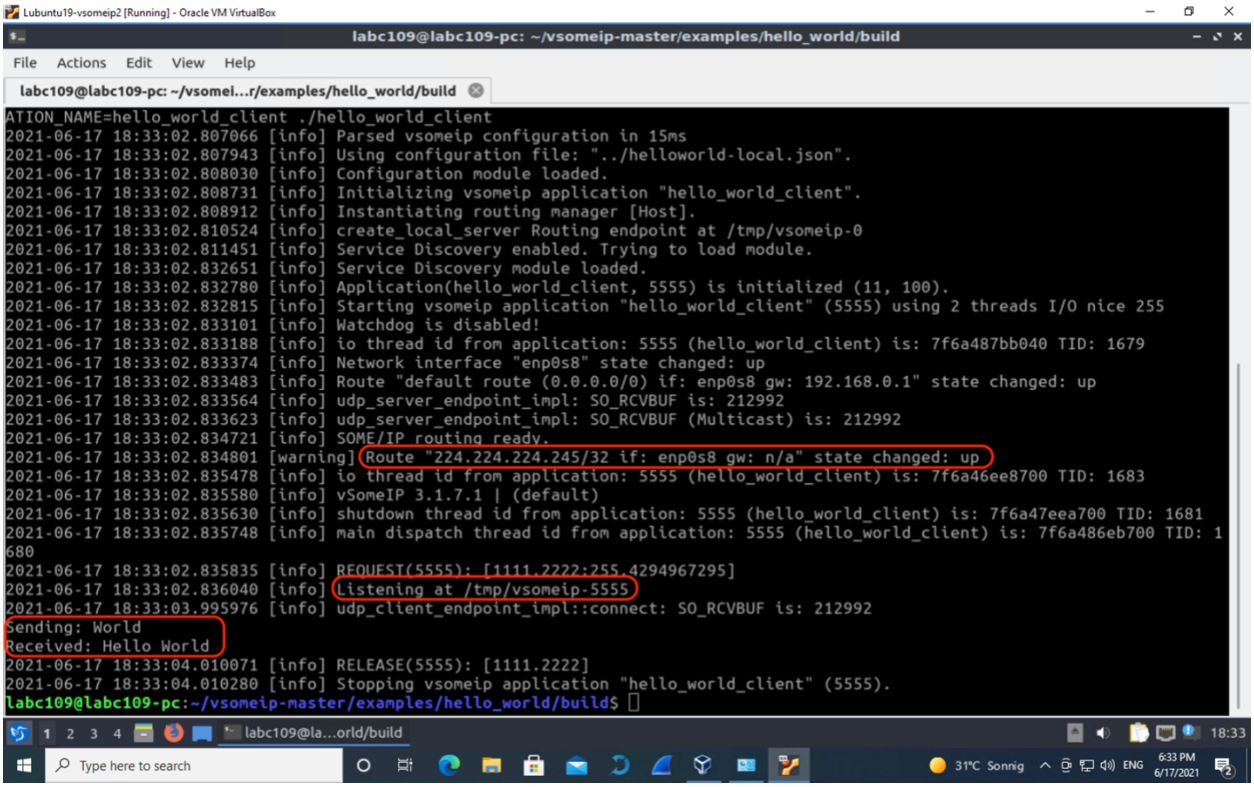
```
VSOMEIP_CONFIGURATION=../helloworld-local.json \
VSOMEIP_APPLICATION_NAME=hello_world_client \
./hello_world_client
```

The above commands make sure to run the specific application reading our custom configurations from the JSON file.

Service Side:



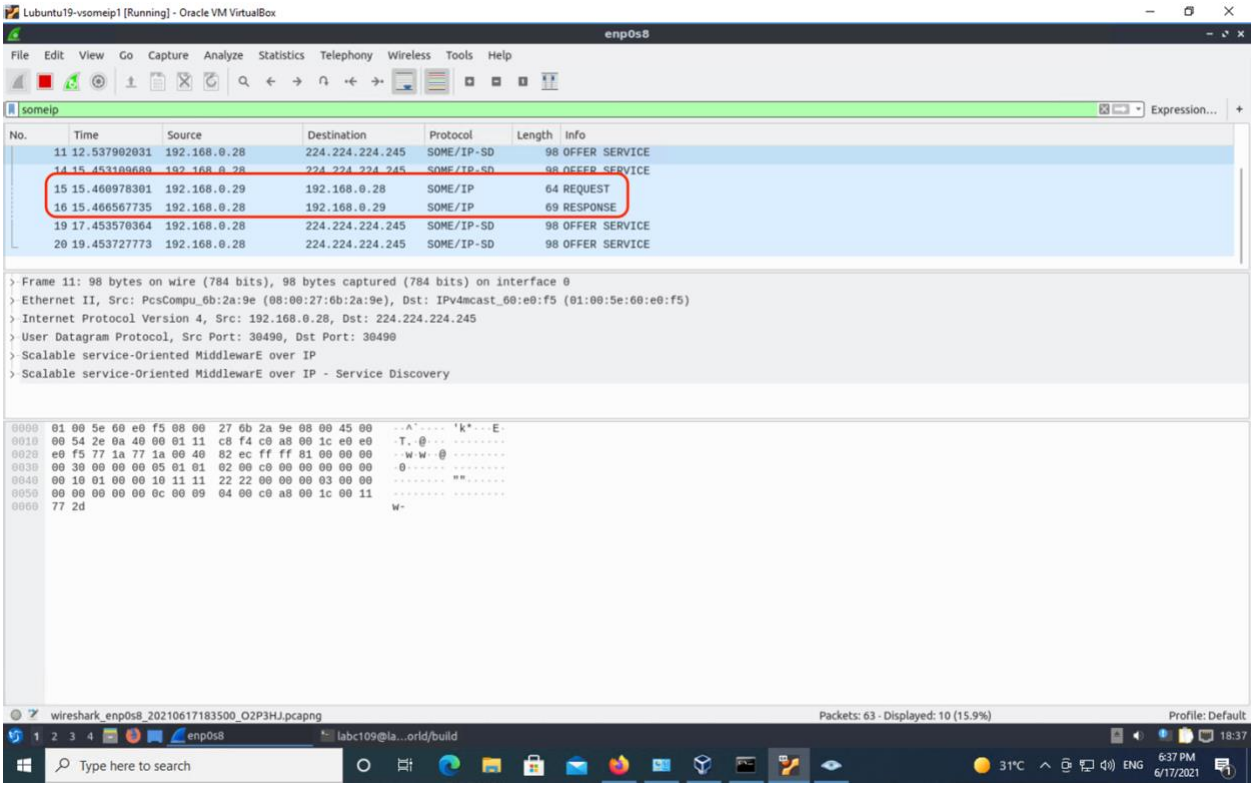
Client Side:



As we can see we got our response back ‘Hello World’ and after that the connection closed for both nodes.

STEP 4: ANALYZING TRAFFIC USING WIRESHARK

We use Wireshark to analyze data traffic. The figure below shows the data received from PC1. This image shows multiple PC1 service offerings broadcast. There are also "REQUEST" and "ANSWER" messages. The REQUEST message is sent from the client to the service node, and the RESPONSE message is sent from the service node to the client node. For a better overview, we put the "someip" filter in the "Wireshark" filter section.



Details about the "RESPONSE" message we can see the received information:
• Source port and destination port-"30490", used to discover services through the "UDP" protocol.
• "MessageID" is composed of the identifiers "Service" and "Method", here it is "0xffff810". This is only used for service identification.
• The values of "ProtocolVersion" and "InterfaceVersion" are "0x01", which are used to find services.
• "MessageType" is the "Notification" with a value of "0x02".
• "ReturnCode" is the "E_OK" has the value "0x00", which means that no error occurred.

Details about the "REQUEST" message we can see the sent information:

• The source IP address is 192.168.29 (PC2, client node), and the destination IP address is 192.168.28 (PC1, service)
• The source port is 38647, used for the client node, port the target is 30509 for the service host.
• Here, the service ID we use for the service is 0x1111, and the method ID we use for the client node is 0x3333.
• The protocol version is the same as before (0x01), but here is the interface version (0x00).
• The message type here is request (0x00), and the return code is "E_OK" (0x00), that is, there is no error.
• The data (world) is sent with the payload.

STEP 5: EXAMPLE OUTPUT

After we successfully exchange data between two Linux computers in a virtual machine on a hardware device, we want to use two different real devices.

In our project, we use "Lubuntu19-vsomeip1", which is located on the first computer with an IP address of "192.168.28" as a host "service", and "Lubuntu19-vsomeip2" with an IP address of "192.168.29" as a "client" on the second computer.

The two computers are connected to each other through a media gateway. The media gateway is like a simple switch. We use the default settings of the media gateway.

Data traffic runs through the media gateway, The two nodes communicate with each other via SOME/IP. No need to change any setting code or C++ code, the startup and analysis process is exactly the same as before. The only difference between REQUEST and RESPONSE information obtained using Wireshark is that they use different source and destination ports for communication. The following is the information we received from Wireshark's REQUEST message. Because the request message comes from a different computer, it uses a different font number. The rest of the information is intact.

So, what can we demonstrates after we ran the example is that the communication between PC1 and PC2 VMs can be described as following:

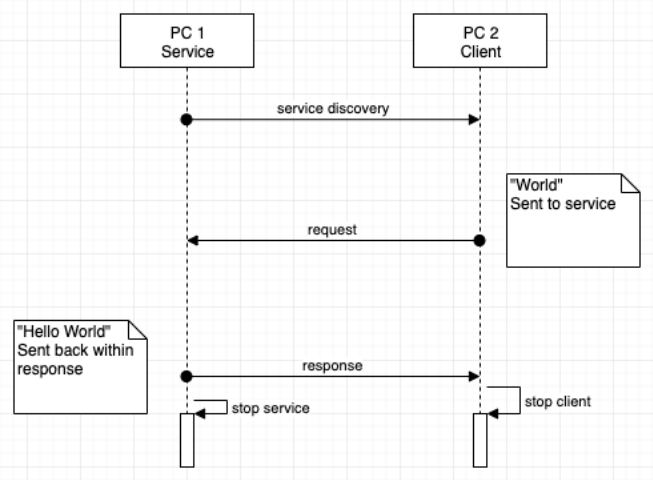


Figure 33: "hello_world" Communication Sequence Diagram

Reference: ***SOME/IP und automobiles Ethernet***

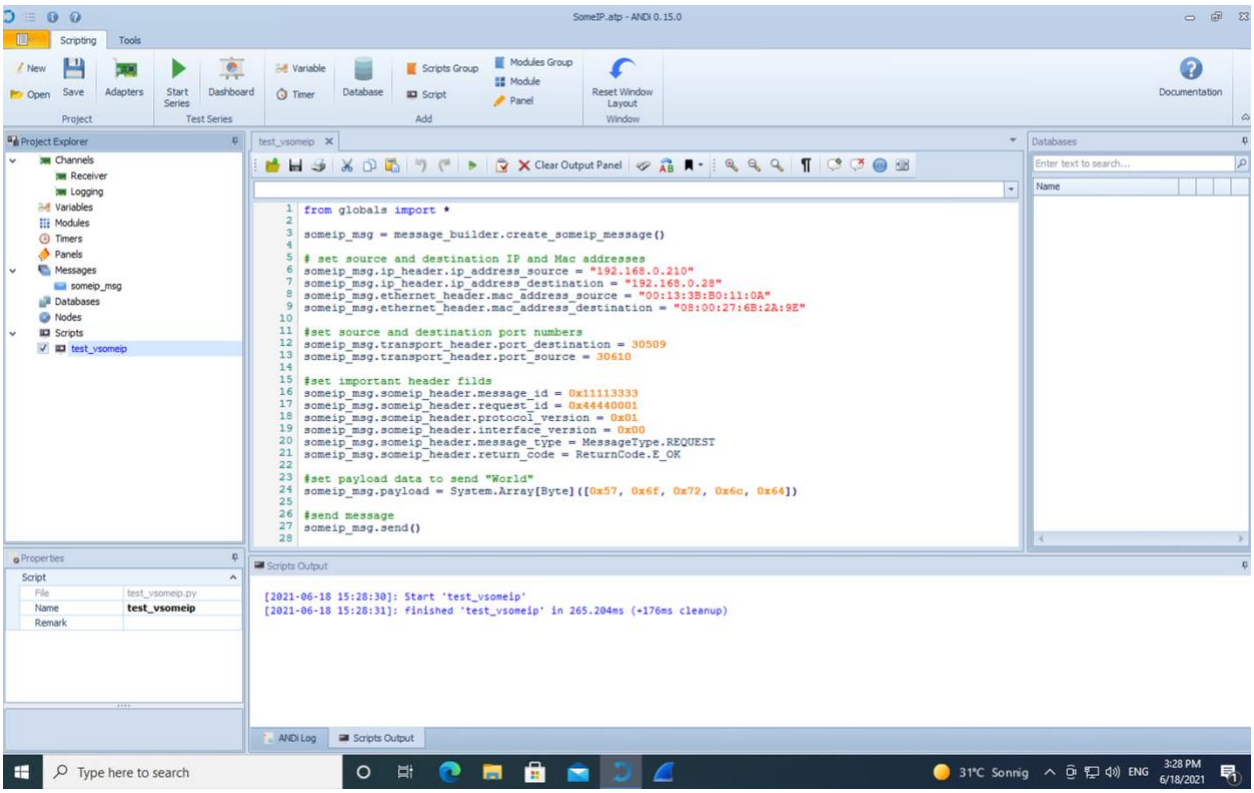
STEP 6: COMMUNICATE USING ANDI

The next task is to model one of the nodes using ANDi. Since Some/IP is a protocol that can be used on different operating systems, we need to be able to communicate between Linux computers and Andi on Windows. In the project, we use "Lubuntu19-vsomeip1" again, which acts as a "service" node on the first computer with an IP address of "192.168.28" and calculates on our workstation with an IP address of "192.168.0.210" Two (Ethernet1 connected to the media gateway) are used as client nodes.

First, we have to create the ‘someip’ message and then we set the source and destination IP and MAC addresses.

After that we set the source and destination port numbers. Here we can use "30610" as the source port and the destination port number is the same as before "30509" which was used in the service method.

Then we need to set the request id which is her "6666".



Now if we analysis the request through Wireshark we should get the following

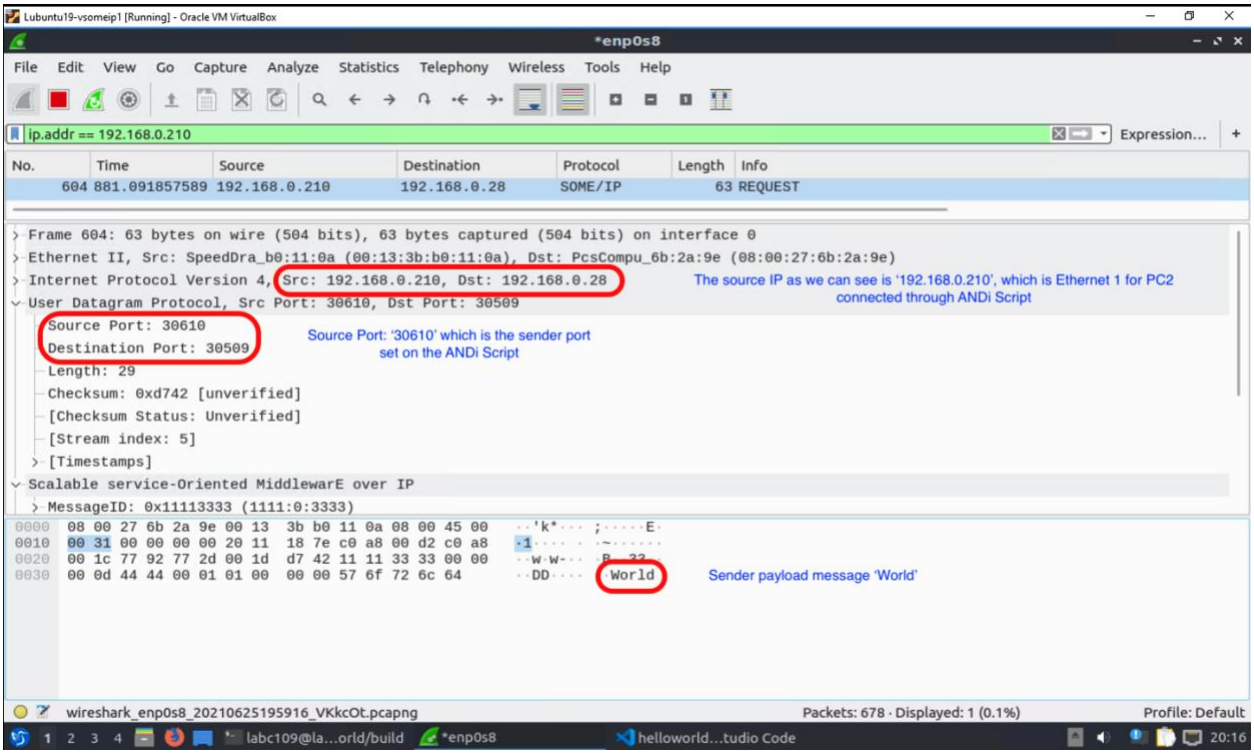


Figure 34: Wireshark on VM1-PC1 for ANDi script

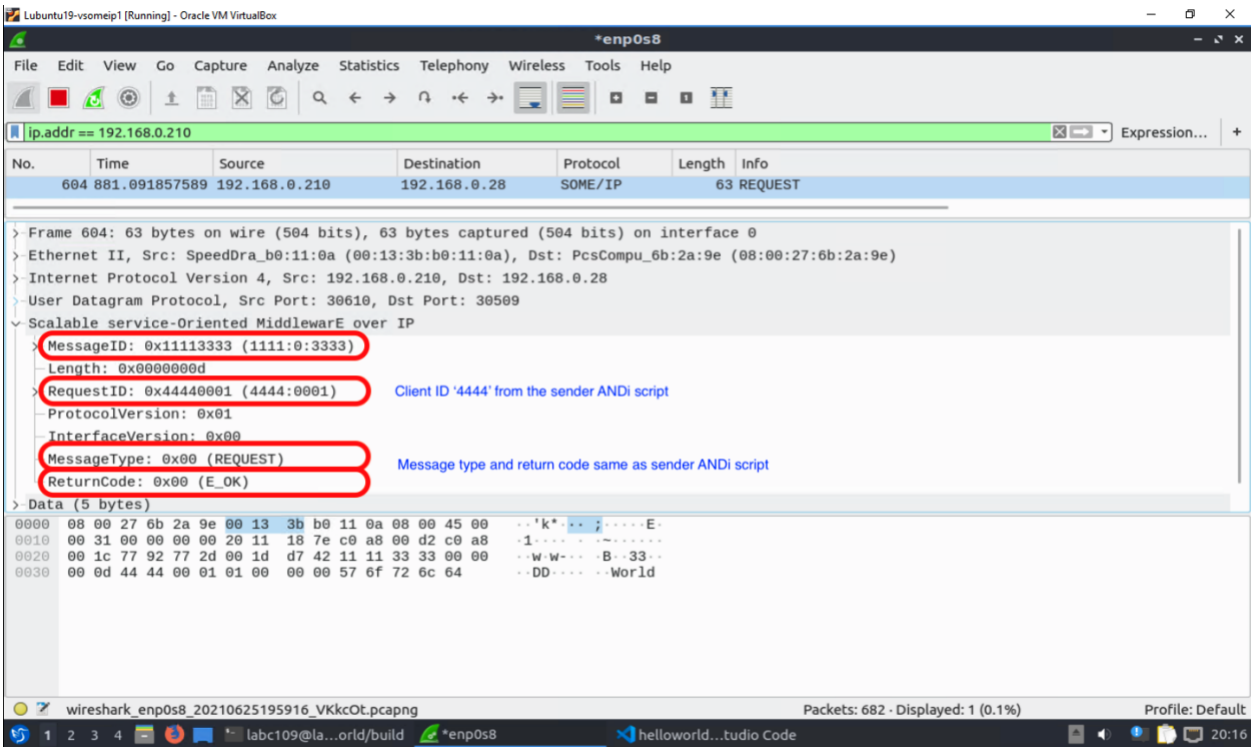


Figure 35: Wireshark on VM1-PC1 for ANDi script

Here we can see that the source IP is "192.168.0.210" from ANDi, and the destination IP is "192.168.0.28", which belongs to Linux on the first computer. In addition, the source port is "30610" we set in the script, and the destination port is "30509" as before. As we set in the script, "RequestID" is "0x44440001". My data has been sent. How do we set the load in the script. The rest of the information is the same as before.

CONCLUSION

In this semester, we learned a lot by interacting with four different interrelated tasks. In fact, every task is a stepping stone for the next task. In the first task, we will start with two simple Python sending and receiving scripts, which will interact with the loopback adapter. This helped us understand how ANDi communication works, and we learned how to use different tools and analyze Wireshark data. In the second task, we use our script in the real network and start communicating and analyzing with "Wireshark".
The next step is to configure the VLAN with the media gateway to create a virtual network within the network, we learned how to set up a media gateway and how to set the correct ID for each port and switch. If the configuration is correct, we can communicate with the devices in the VLAN and analyze the recorded data.
We even connected one of our devices to a camera connected to a media gateway.In the third task, we further interact with other devices in the network infrastructure. There is a central media gateway to connect multiple workstations and a central camera.
With the correct port configuration, we can connect to the headquarters and connect the camera and Analyze the recorded data.

In the fourth task, the project is very difficult because it requires the various skills and materials we learned in this semester to communicate with the protocol called "VSomeIP", which can be used to build Ethernet-based automotive equipment (as cameras, ..) which they are often used by various operating systems. In this exercise, we learned how to use smoeip to configure, start, and interact with two devices with the same operating system (Linux) and different operating systems (Windows). We learned how to use ANDi to model one of our nodes (clients). This node on Windows can communicate with another node (service) on Linux, and we can analyze data from Wireshark. In general, this workshop is a good experience for learning new things and solving problems.

So, in the end, we would like to thank Prof. Dr. Ing. Andreas Grzemba for the material was provided through the course and Ing. Johann Bretzendorfer for his help during our labs.
Because we believe that this course gave us the good knowledge to dig in more on the automotive field because as we all know this field is very important and became a main part in the current century.