

# NWEN302

## Lab 1

Olivia Fletcher  
300534281  
fletcholiv

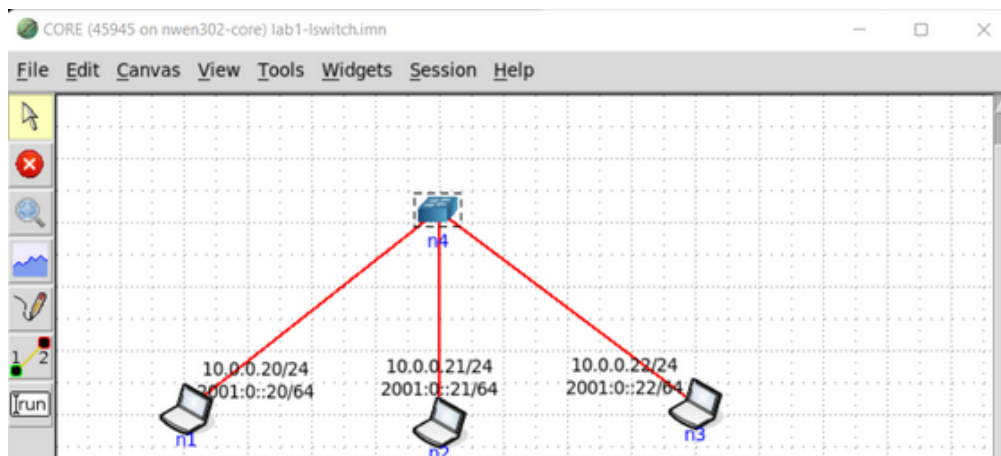
## NPD Address Solution

### Introduction

The NPD, Neighbor Discovery Protocol is a protocol operated within the link layer of the OSI model - internet protocol suite. The link layer is responsible for communications related to the link that a host is connected to. In this report we will be discerning the two main concepts of NPD, one of which being; Address Resolution which is responsible for multiple-address access network functioning. Another key feature and resource we will be assessing is the use of learning switches. Switches are used as a means to ensure traffic is only sent to the destination IP rather than every device on the network. We will be implementing a 'learning switch' which is the process of obtaining the MAC of all devices to a connected network and comparing it to the MAC addresses stored on the switch table.

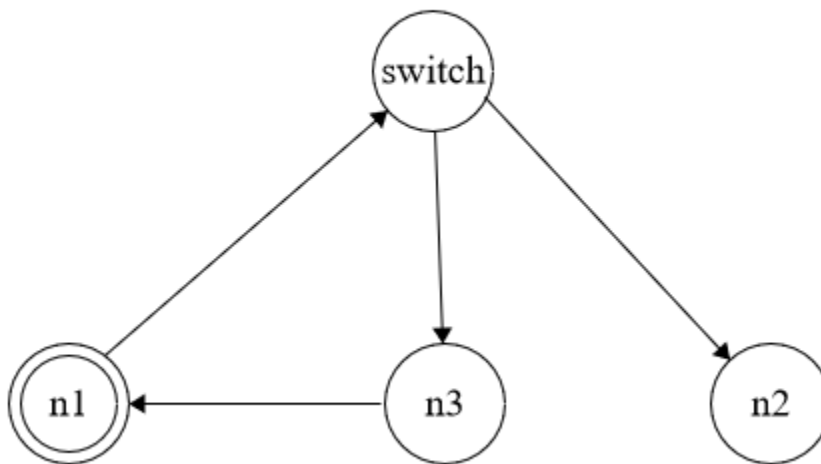
### Design

We will be utilizing core GUI inbuilt into the Linux OS, the GUI output is a blank canvas that we can use to create network structure within the virtual machine. We can make use of the guis network setup to implement our own algorithms.



With the correct implementation a node will broadcast through the switch to each of the other nodes and add them to the nodes ARP table, this will allow for an ARP packet to be sent and received from only the destination address and not every other node on the network.

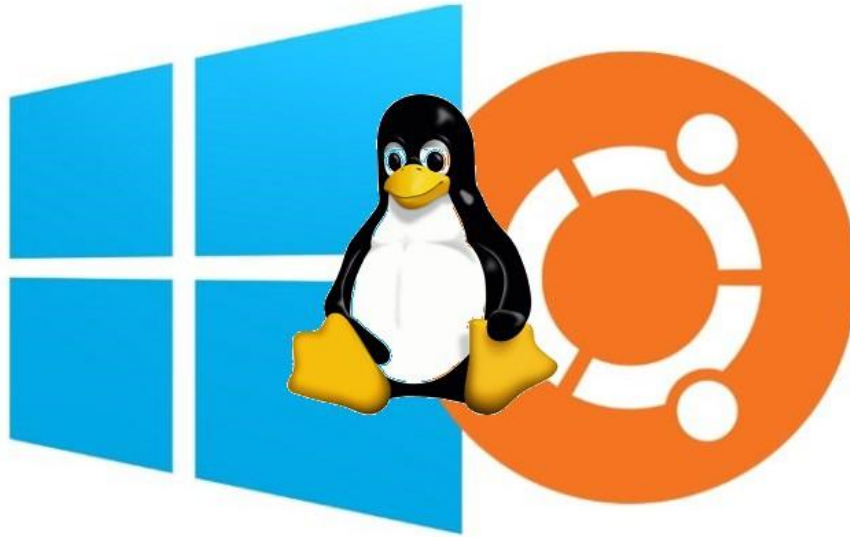
Below is an example of a successful ARP delivered and received packet, If we were to send an ARP packet from node 1 to node 3 but we do not know node 3's MAC address yet the switch will send a broadcast message to each node. Once we have found the MAC associated with the destination (node 3's) IP address we add it to our table and successfully send the ARP packet to the destination.



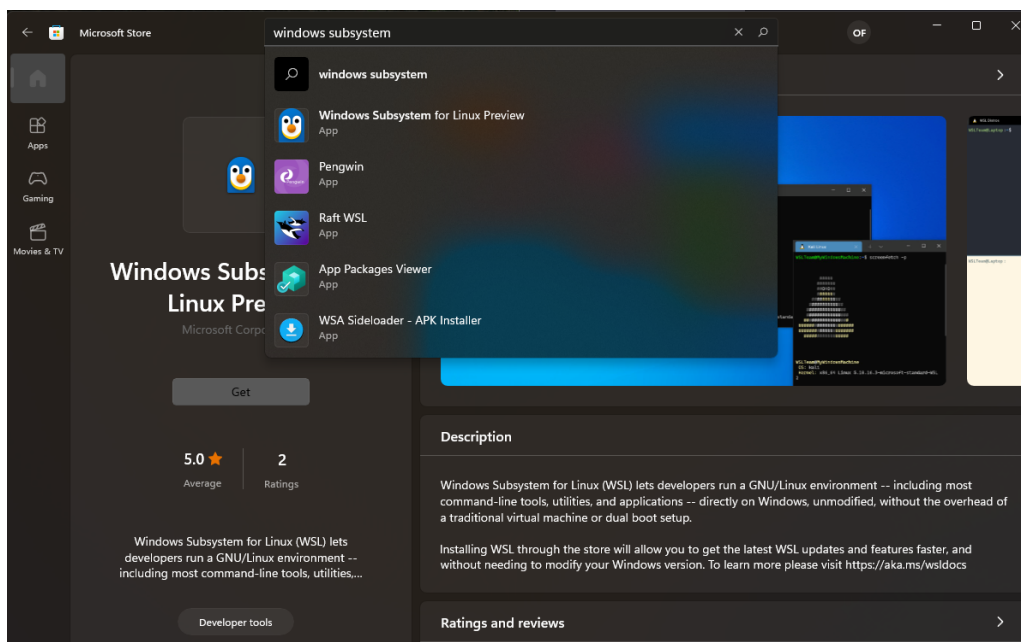
In regards to the learning switch algorithm we will employ a similar design pattern but instead of sending/receiving arp packets we will be evaluating the frames that the switch receives and adding the device data to the kernel switch table, this is how a node will “learn” another node's MAC address.

## Procedures

The software used to complete this investigation was; WSL, WinSCP, PuTTY and VirtualBox VMS. The setup necessary for completing this investigation are such as;



- As my default device OS is Windows I needed to configure the Linux subsystem to allow for Linux capabilities. To set up WSL linux onto our system we need to install WSL offered in the Windows app store.

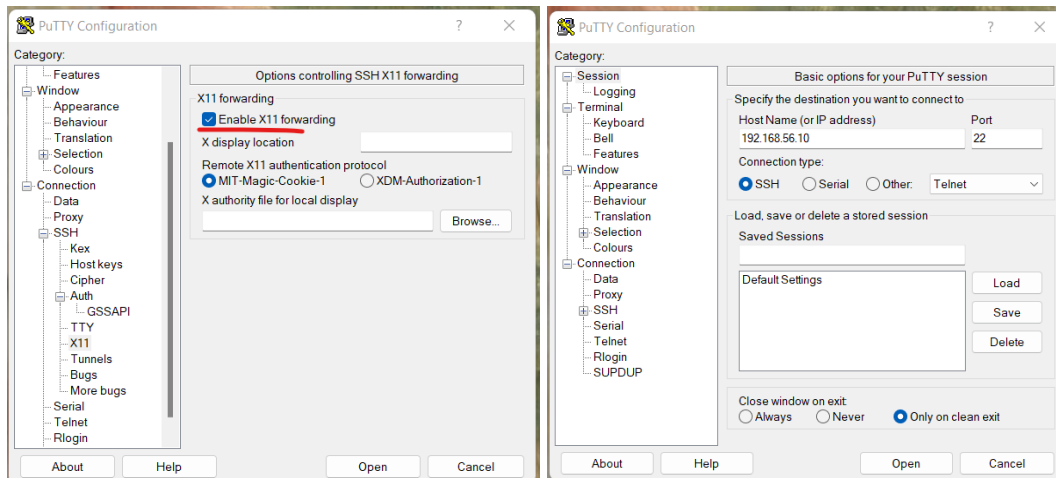


Now with WSL fully implemented onto our windows device we can now use the Linux command line to complete the rest of the investigation.



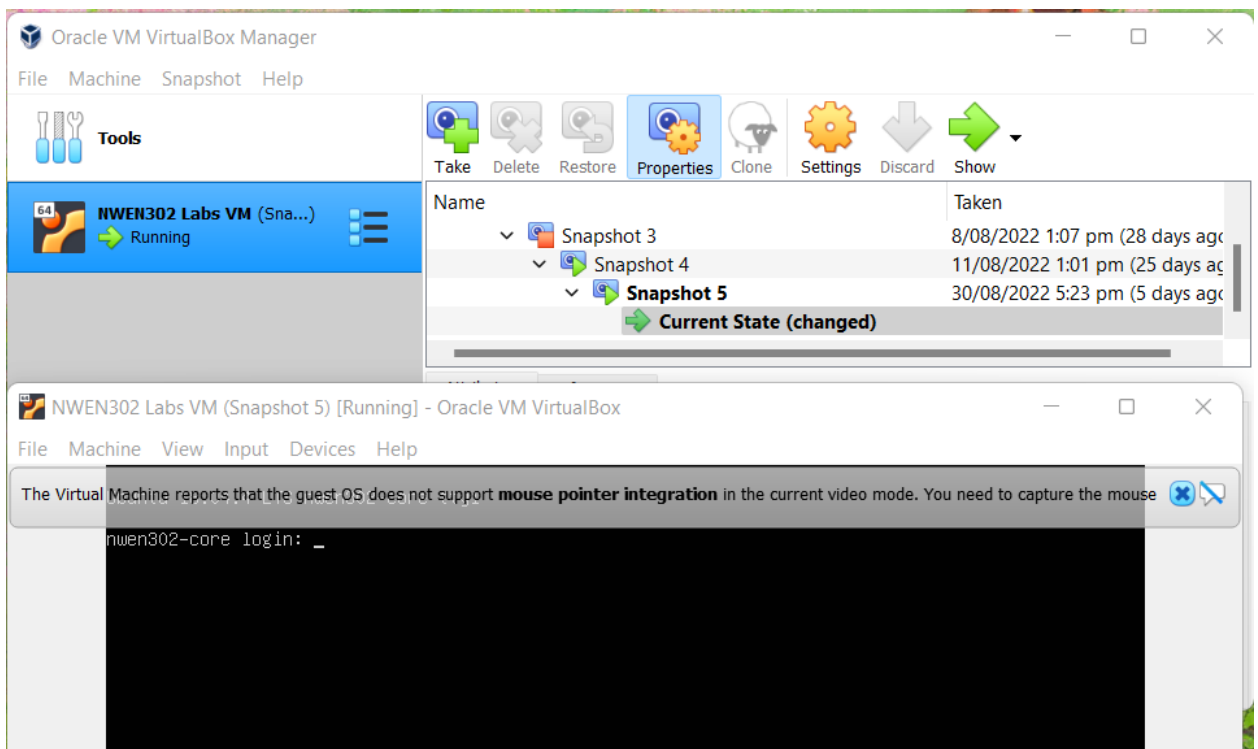
# PuTTY

- PuTTY, allows for remote SSH session capabilities, and we will be using this as our terminal to connect to the network simulation. Only needed settings are; checking “Enable X11 forwarding” in the X11 category under SSH in the menu bar so we can connect to the host address; (using SSH, port 22) 192.168.56.10





- VirtualBox VMs is a tool used for creating a virtualized environment, this will allow for us to extend our existing device to run another OS. We will be importing an already set image for the VM to base our work on, once starting the machine a terminal will open with a logon screen, at this point we can now connect to the VM in a separate terminal using PuTTY. Once connected with Putty we can begin the investigation with our code based algorithms running through the GUI screen. Below shows the running VM.



At the moment of running the machine we will open a PuTTY session, connect to the VM and begin implementing our algorithms.

CORE (40855 on nwen302-core) lab1-lswitch.imn

File Edit Canvas View Tools Widgets Session Help

```
graph TD
    n4[n4] --- n1[n1]
    n4 --- n2[n2]
    n4 --- n3[n3]
```

n1 10.0.0.20/24  
2001:0::20/64

n2 10.0.0.21/24  
2001:0::21/64

n3 10.0.0.22/24  
2001:0::22/64

n4

nwen302@nwen302-core: ~/lab1

```
-- William Shakespeare, "Julius Caesar"

Last login: Tue Aug 30 04:24:39 2022 from 192.168.56.1
nwen302@nwen302-core:~$ ls
lab0 lab1
nwen302@nwen302-core:~$ cd lab1
nwen302@nwen302-core:~/lab1$ ls
arp lab1-net2.imn
core lab1-net2-node2.pcapng
lab1-n4-eth0.pcapng lab1-net2-node3.pcapng
lab1-n4-eth1.pcapng lab1-net2-usingMyArp-node2.pcapng
lab1-n4-eth2.pcapng lab1-net2-usingMyArp-node3.pcapng
lab1-net1.imn lab1-net2-usingMyCode.imn
lab1-net1-node2.pcapng lswitch
lab1-net1-node3.pcapng nwen302-lab1-1.2.tar.gz
nwen302@nwen302-core:~/lab1$ sudo core-gui
[sudo] password for nwen302:
Connecting to "core-daemon" (127.0.0.1:4038)...connected.
Connection to "core-daemon" (127.0.0.1:4038) closed.
Connecting to "core-daemon" (127.0.0.1:4038)...connected.
```

Canvas1

zoom 100%

Address resolution pseudo code;

```
1  void my_arp_resolve (uint32_t ip_address) {
2
3      // Create a char array (for the destination MAC) with a set size of 6
4      // Set the destination MAC array to the broadcast address as seen below;
5
6      // {0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
7
8      // Call the arp_send_query function from the arp_functions class passing in
9      // the ip_address parametre stored in this method and passing in our created
10     // broadcast address
11
12     // Set the method call to an int for later use
13
14     // If the arp_send_query function call is equal to 0
15     // Add a print statement for error checking
16     // Otherwise, successfully broadcasted !
17 }
18
19 void my_arp_handle_request(uint32_t ip_address, const unsigned char *mac_address) {
20
21     // Create an empty chararray for the buffer and set to 1000
22     // Create another array specifically for our MAC address and set size to 6
23
24     // Call the arp_get_my_ipaddr function from the arp_functions class
25     // and set to a char variable for our ip_address needed for later use
26
27     // Now -
28     /* Compose Ethernet Header */
29
30     /* Compose Arp Header */
31
32     /* Filling in Contents */
33
34     /* Reply */
35     // Use the created structures from the ethernet header, arp header and arp packet contents
36     // Send a reply using the arp_send_reply with the eth frame and size of the each of
37     // above structures
38
39 }
40
41 void my_arp_handle_reply(uint32_t ip_address, const unsigned char *mac_address) {
42     // Call the arp_insert_cache function with the above parametres
43 }
```

Learning switch pseudo code;

```
1 // Required field;
2 // int value for our Interface data
3 // int value for our Position data needed to assign the position in the structure
4
5 // Create a structure used for our kernel entries with;
6 // A source_mac address pointer
7 // A interface pointer
8 // A timer function
9
10 // Create an array with the structure and set to a size of 3 (one for each device)
11
12 void my_lswitch_frame_receive(const unsigned char *source_mac_address,
13     const unsigned char *dest_mac_address, const char *device) {
14
15     // Created a "found" boolean and set to false for later use
16     // Create a copy of the entry structure for temporary use
17
18     // Debugging,
19     // Create a copy of the parametre source_mac_address and use memory allocation
20     // to take the source_mac_address data into our copies version
21
22     // Set the copied MAC & device info to the temp structure
23
24     // Loop through the size of the structure (3) {
25     // Compare the current item in the tabel to the one stored on our
26     // temporary structure {
27     // If the items are the same, set the found boolean to 'true' }}
28
29     // Using the found boolean check if the table is empty {
30     // If the table contains no items add entries to the table using the
31     // lswitch_insert_table function call with our tempStruct data
32     // Increment the position variable }
33 }
34
35 void expired_arp_entries(const unsigned char *source_mac_address, const char *device) {
36     // Using the entry struct data compare items within in the structure and set
37     // a timer once an entry has been made, after 30 seconds remove the entry from
38     // the table
39 }
```

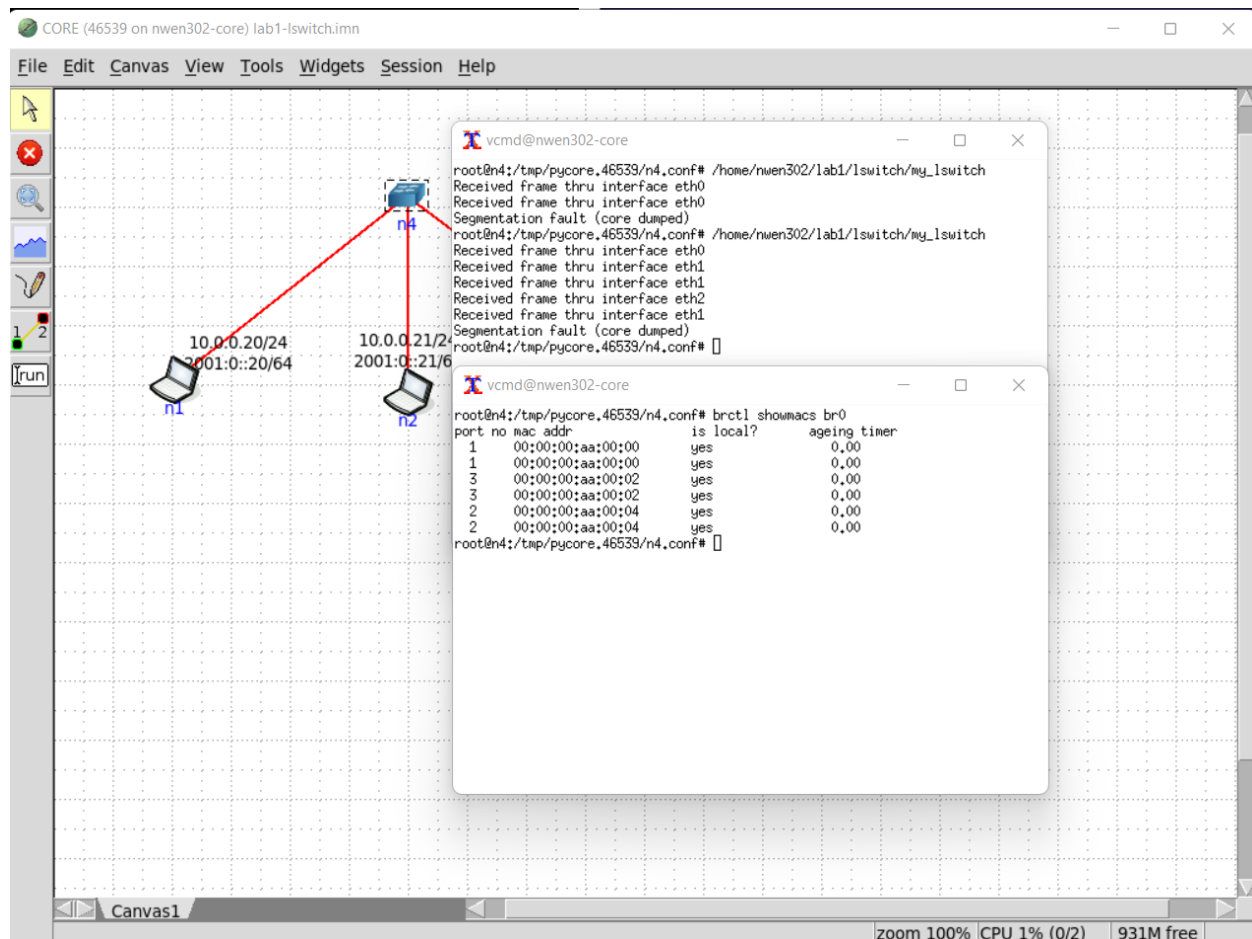
With my switch algorithm I created an empty instance of structure which will be used to store arp entries. The function itself is quite simple, only being a for loop which checks each position in the array against the data stored within the table, then adds a kernel to the table position that is free.



## Results

In order to resolve address resolution in neighbor discovery we successfully implemented our own ARP algorithm which is able to obtain a target's MAC address by broadcasting to each node in the network and adding it to the ARP cache table. Once the target MAC address has been found only then can the node successfully be able to handle the ARP request and correct ARP reply functionality. The results show that every node receives the initial broadcast request while only the destination and source nodes receive the packet data.

When compiling my code for the switch I didn't encounter any errors with the compiler itself but when I went to run the code through the network simulation I encountered a "segmentation fault".



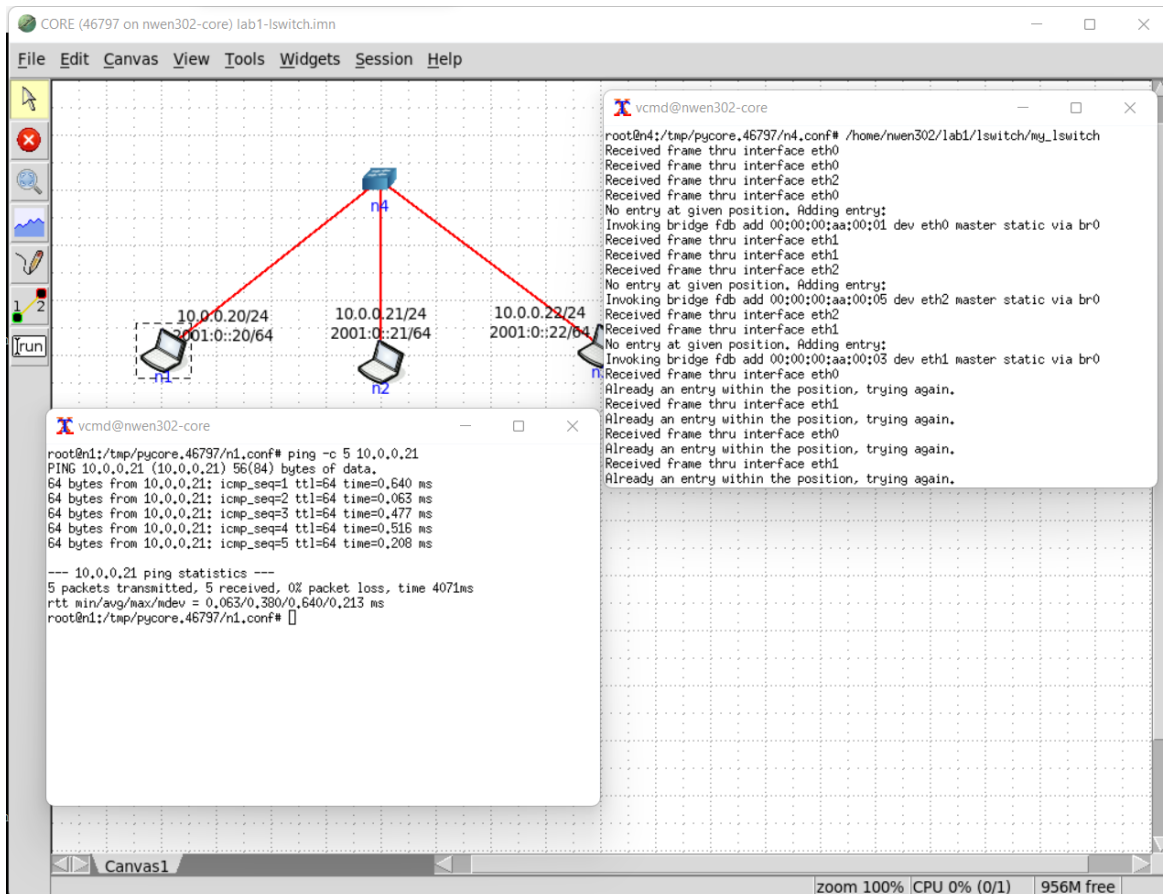
From resources online, a segmentation fault could be due to a memory allocation error. When debugging my code from below I believe it is to do with my for loop where inside I compare the temporary structures entry to the one at the looped position.

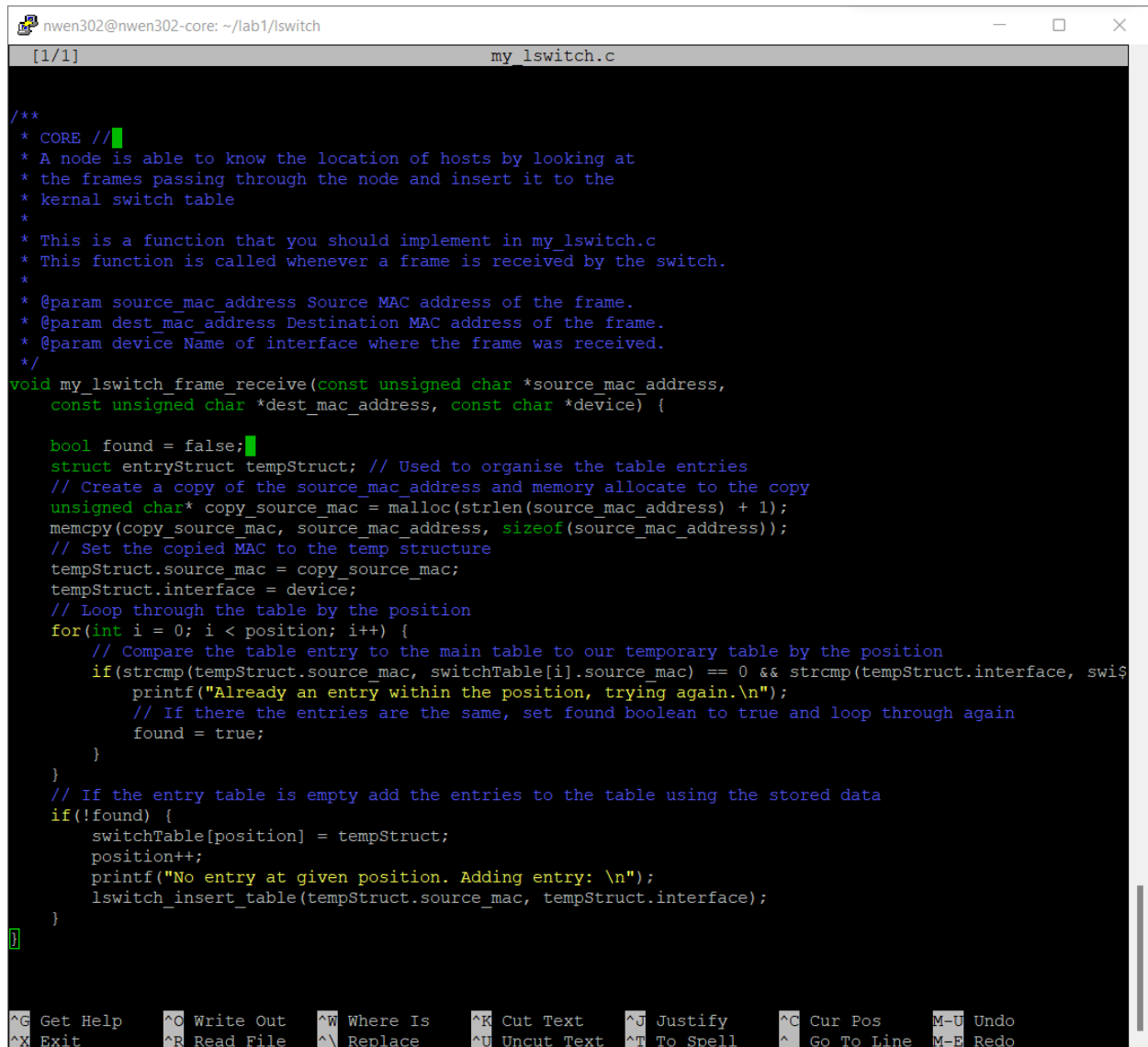
```

56 void my_lswitch_frame_receive(const unsigned char *source_mac_address,
57     const unsigned char *dest_mac_address, const char *device) {
58     bool found = false;
59     struct entryStruct tempStruct;
60     unsigned char* copy_source_mac = malloc(strlen(source_mac_address) + 1);
61     memcpy(copy_source_mac, source_mac_address, sizeof(source_mac_address));
62     // Set the copied MAC to the temp structure
63     tempStruct.source_mac = copy_source_mac;
64     tempStruct.interface = device;
65     // if array empty add items
66     for(int i = 0; i < 3; i++) {
67         // get the source mac within our struct
68         if(strcmp(tempStruct.source_mac, switchTable[i].source_mac) == 0
69             && strcmp(tempStruct.interface, switchTable[i].interface) == 0) {
70             found = true;
71         }
72     }
73     // If the entry table is empty add the entries to the table using the stored data
74     if(!found) {
75         printf("here");
76         switchTable[position] = tempStruct;
77         position++;
78         lswitch_insert_table(tempStruct.source_mac, tempStruct.interface);
79     }
80 }

```

To fix this, I changed the loop so that it's maximum counts to the position variable, not the "3". This ensures that if it goes back through at the same position it doesn't mess up the memory allocation of the string compare statement. When running the code with this fix we no longer get a segmentation fault:





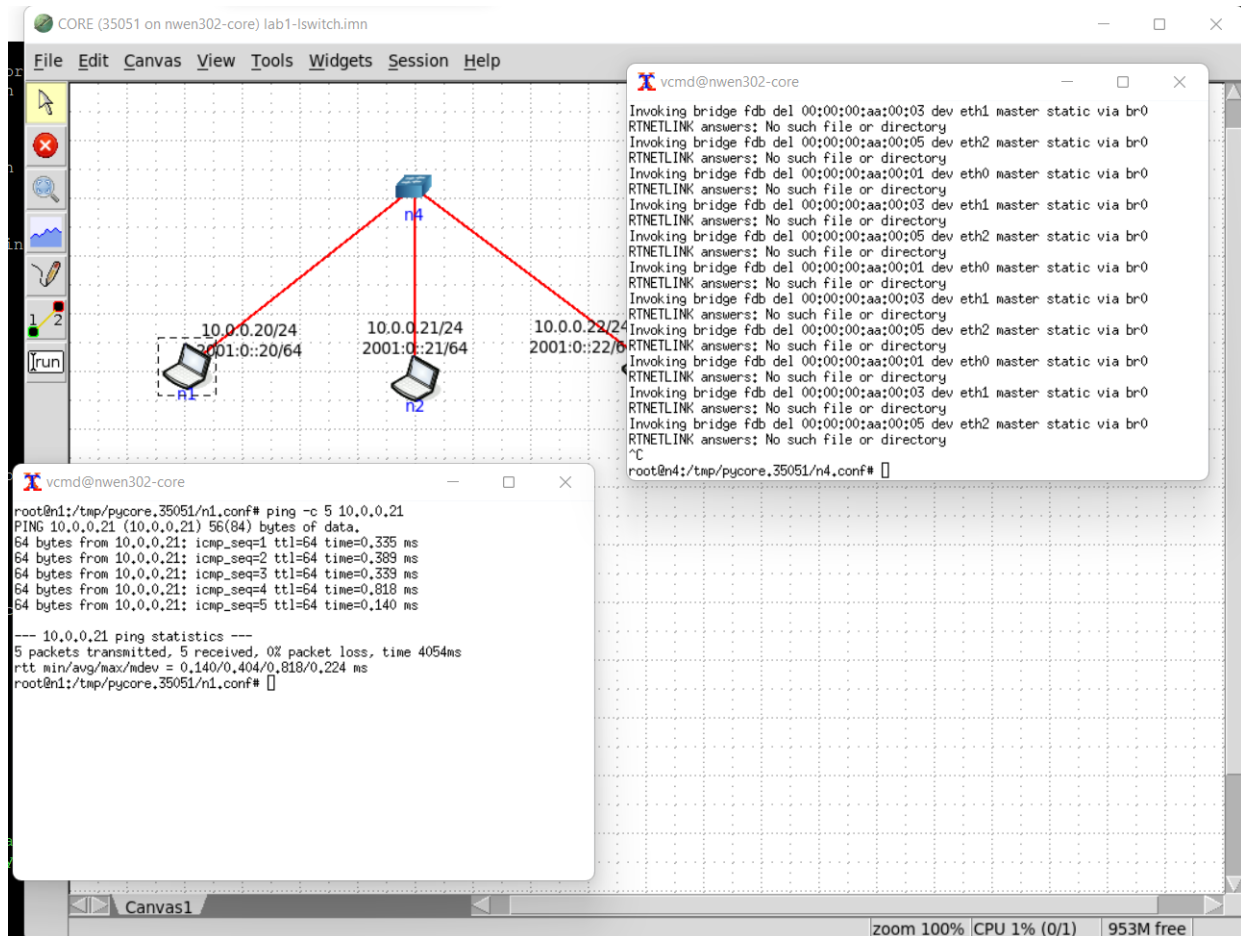
```
nwen302@nwen302-core: ~/lab1/lswitch
[1/1] my_lswitch.c

/**
 * CORE //
 * A node is able to know the location of hosts by looking at
 * the frames passing through the node and insert it to the
 * kernal switch table
 *
 * This is a function that you should implement in my_lswitch.c
 * This function is called whenever a frame is received by the switch.
 *
 * @param source_mac_address Source MAC address of the frame.
 * @param dest_mac_address Destination MAC address of the frame.
 * @param device Name of interface where the frame was received.
 */
void my_lswitch_frame_receive(const unsigned char *source_mac_address,
                             const unsigned char *dest_mac_address, const char *device) {

    bool found = false;
    struct entryStruct tempStruct; // Used to organise the table entries
    // Create a copy of the source_mac_address and memory allocate to the copy
    unsigned char* copy_source_mac = malloc(strlen(source_mac_address) + 1);
    memcpy(copy_source_mac, source_mac_address, sizeof(source_mac_address));
    // Set the copied MAC to the temp structure
    tempStruct.source_mac = copy_source_mac;
    tempStruct.interface = device;
    // Loop through the table by the position
    for(int i = 0; i < position; i++) {
        // Compare the table entry to the main table to our temporary table by the position
        if(strcmp(tempStruct.source_mac, switchTable[i].source_mac) == 0 && strcmp(tempStruct.interface, swi$
            printf("Already an entry within the position, trying again.\n");
            // If there the entries are the same, set found boolean to true and loop through again
            found = true;
        }
    }
    // If the entry table is empty add the entries to the table using the stored data
    if(!found) {
        switchTable[position] = tempStruct;
        position++;
        printf("No entry at given position. Adding entry: \n");
        lswitch_insert_table(tempStruct.source_mac, tempStruct.interface);
    }
}

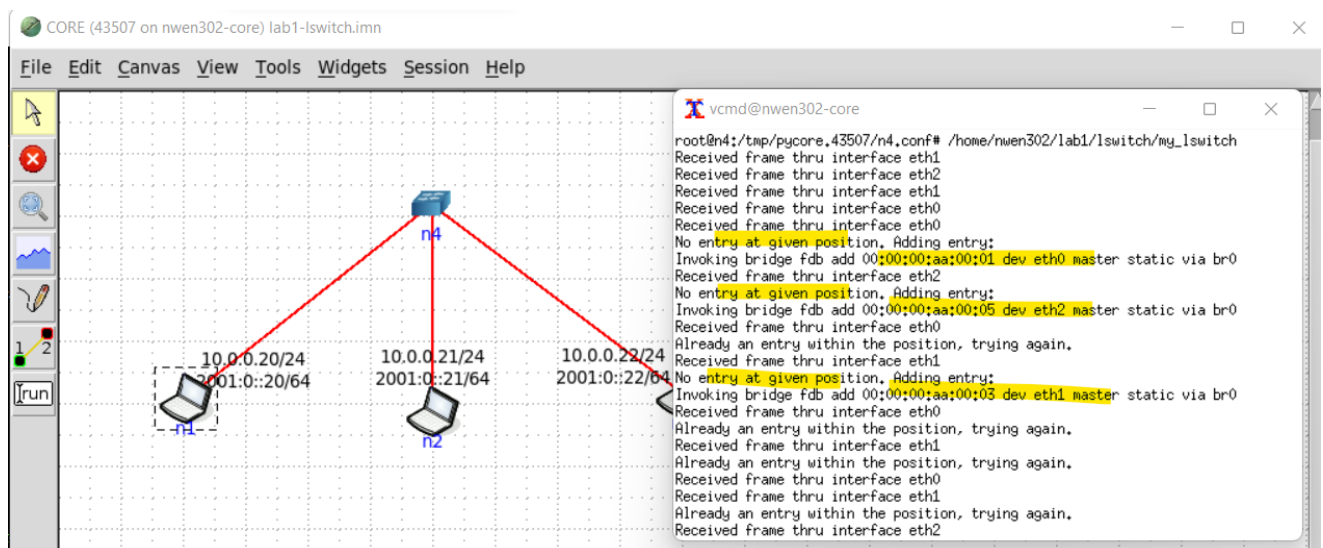
^G Get Help      ^O Write Out    ^W Where Is     ^R Cut Text     ^J Justify      ^C Cur Pos     M-U Undo
^X Exit          ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Spell     ^ Go To Line   M-E Redo
```

My next step was to implement a timer function which checks the age of each entry on the table and removes it after 30 seconds. I implemented the `time_t` which uses an `alarm()` function, the way I structured it was just a simple for loop and if statement which compares the current entries age on the table to the started timer. Once the difference is 30 seconds, call the `lswitch_delete_table` function provided. I believe this is the correct implementation but as we can see in the screenshot below, we need to also include a way where the program records what the entries position was when it was deleted.



The “RTNETLINK answers: No such file or directory” is due to a problem with the kernel modules due to the above reason.

When commenting out the `lswitch_delete_table` function the simulation works as follows:



Successfully learning (adding to table) but not resetting and removing kernels.

The brctl showmacs br0 before running the simulation:

```

vcmd@nwen302-core
root@n4:/tmp/pycore.43507/n4.conf# brctl showmacs br0
port no mac addr is local? ageing timer
1 00:00:00:aa:00:00 yes 0,00
1 00:00:00:aa:00:00 yes 0,00
2 00:00:00:aa:00:02 yes 0,00
2 00:00:00:aa:00:02 yes 0,00
3 00:00:00:aa:00:04 yes 0,00
3 00:00:00:aa:00:04 yes 0,00
root@n4:/tmp/pycore.43507/n4.conf#

```

After:

The screenshot displays a network simulation interface with three nodes (n1, n2, n3) connected to a central switch (n4). The nodes are represented by laptop icons with their IP addresses and subnets: n1 (10.0.0.20/24), n2 (10.0.0.21/24), and n3 (10.0.0.22/24). The switch n4 is represented by a blue square icon. The terminal windows show the brctl showmacs br0 command output, which now includes entries for node 1 and node 2's communication. The terminal for node n1 shows a ping command being executed to node 2.

```

vcmd@nwen302-core
root@n4:/tmp/pycore.43507/n4.conf# brctl showmacs br0
port no mac addr is local? ageing timer
1 00:00:00:aa:00:00 yes 0,00
1 00:00:00:aa:00:00 yes 0,00
2 00:00:00:aa:00:02 yes 0,00
2 00:00:00:aa:00:02 yes 0,00
3 00:00:00:aa:00:04 yes 0,00
3 00:00:00:aa:00:04 yes 0,00
root@n4:/tmp/pycore.43507/n4.conf# brctl showmacs br0
port no mac addr is local? ageing timer
1 00:00:00:aa:00:00 yes 0,00
1 00:00:00:aa:00:00 yes 0,00
2 00:00:00:aa:00:02 yes 0,00
2 00:00:00:aa:00:02 yes 0,00
3 00:00:00:aa:00:04 yes 0,00
3 00:00:00:aa:00:04 yes 0,00
root@n4:/tmp/pycore.43507/n4.conf# brctl showmacs br0
port no mac addr is local? ageing timer
1 00:00:00:aa:00:00 yes 0,00
1 00:00:00:aa:00:00 yes 0,00
2 00:00:00:aa:00:02 yes 0,00
2 00:00:00:aa:00:02 yes 0,00
3 00:00:00:aa:00:04 yes 0,00
3 00:00:00:aa:00:04 yes 0,00
root@n4:/tmp/pycore.43507/n4.conf#

```

```

vcmd@nwen302-core
root@n1:/tmp/pycore.43507/n1.conf# ping -c 5 10.0.0.21
PING 10.0.0.21 (10.0.0.21) 56(84) bytes of data:
64 bytes from 10.0.0.21: icmp_seq=1 ttl=64 time=0.746 ms
64 bytes from 10.0.0.21: icmp_seq=2 ttl=64 time=0.227 ms
64 bytes from 10.0.0.21: icmp_seq=3 ttl=64 time=0.145 ms
64 bytes from 10.0.0.21: icmp_seq=4 ttl=64 time=0.157 ms
64 bytes from 10.0.0.21: icmp_seq=5 ttl=64 time=0.072 ms

--- 10.0.0.21 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4054ms
rtt min/avg/max/mdev = 0.072/0.269/0.746/0.243 ms
root@n1:/tmp/pycore.43507/n1.conf# ping -c 10 10.0.0.21

```

This shows the learning is working. On node 1, I sent a ping to node 2 and showed the mac table on the switch had added the entries of node1 and 2's communication (but not 3 as 3 was not included). Below are each interfaces wireshark (from the switch n4)



## Interface 0: (node 1)

Wireshark veth4.0.5a

Filter: `ip || arp`

No.	Time	Source	Destination	Protocol	Length	Info
14	123.273885980	00:00:00 aa:00:01	Broadcast	ARP	42	Who has 10.0.0.21? Tell 10.0.0.20
15	123.274062038	00:00:00 aa:00:03	00:00:00 aa:00:01	ARP	42	10.0.0.21 is at 00:00:00:aa:00:03
16	123.274084476	10.0.0.20	10.0.0.21	ICMP	98	Echo (ping) request id=0x001e, seq=1/
17	123.274333223	10.0.0.21	10.0.0.20	ICMP	98	Echo (ping) reply id=0x001e, seq=1/
18	124.276460911	10.0.0.20	10.0.0.21	ICMP	98	Echo (ping) request id=0x001e, seq=2/
19	124.276605798	10.0.0.21	10.0.0.20	ICMP	98	Echo (ping) reply id=0x001e, seq=2/
21	125.280373854	10.0.0.20	10.0.0.21	ICMP	98	Echo (ping) request id=0x001e, seq=3/
22	125.280446123	10.0.0.21	10.0.0.20	ICMP	98	Echo (ping) reply id=0x001e, seq=3/
23	126.304689280	10.0.0.20	10.0.0.21	ICMP	98	Echo (ping) request id=0x001e, seq=4/
24	126.304765470	10.0.0.21	10.0.0.20	ICMP	98	Echo (ping) reply id=0x001e, seq=4/
26	127.328042229	10.0.0.20	10.0.0.21	ICMP	98	Echo (ping) request id=0x001e, seq=5/
27	127.328076885	10.0.0.21	10.0.0.20	ICMP	98	Echo (ping) reply id=0x001e, seq=5/
28	128.513090299	00:00:00 aa:00:03	00:00:00 aa:00:01	ARP	42	Who has 10.0.0.20? Tell 10.0.0.21
29	128.513152904	00:00:00 aa:00:01	00:00:00 aa:00:03	ARP	42	10.0.0.20 is at 00:00:00:aa:00:01

Frame 14: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0  
 Ethernet II, Src: 00:00:00 aa:00:01 (00:00:00:aa:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
 Address Resolution Protocol (request)

0000 ff ff ff ff ff ff 00 00 00 aa 00 01 08 06 00 01 .....  
 0010 08 00 06 04 00 01 00 00 00 aa 00 01 0a 00 00 14 .....  
 0020 00 00 00 00 00 00 0a 00 00 15 .....

wireshark\_veth4.0.5a\_20220912045513\_X9feRo.pcapng Packets: 38 · Displayed: 14 (36.8%) Profile: Default

## Interface 1: (node2)

Wireshark veth4.1.5a

Filter: `ip || arp`

No.	Time	Source	Destination	Protocol	Length	Info
16	123.274456261	10.0.0.20	10.0.0.21	ICMP	98	Echo (ping) request id=0x001e, seq=1/
17	123.274689897	10.0.0.21	10.0.0.20	ICMP	98	Echo (ping) reply id=0x001e, seq=1/
18	124.276858979	10.0.0.20	10.0.0.21	ICMP	98	Echo (ping) request id=0x001e, seq=2/
19	124.276963665	10.0.0.21	10.0.0.20	ICMP	98	Echo (ping) reply id=0x001e, seq=2/
21	125.280769751	10.0.0.20	10.0.0.21	ICMP	98	Echo (ping) request id=0x001e, seq=3/
22	125.280805897	10.0.0.21	10.0.0.20	ICMP	98	Echo (ping) reply id=0x001e, seq=3/
23	126.305087750	10.0.0.20	10.0.0.21	ICMP	98	Echo (ping) request id=0x001e, seq=4/
24	126.305124672	10.0.0.21	10.0.0.20	ICMP	98	Echo (ping) reply id=0x001e, seq=4/
26	127.328421365	10.0.0.20	10.0.0.21	ICMP	98	Echo (ping) request id=0x001e, seq=5/
27	127.328439600	10.0.0.21	10.0.0.20	ICMP	98	Echo (ping) reply id=0x001e, seq=5/

Frame 16: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0  
 Ethernet II, Src: 00:00:00 aa:00:01 (00:00:00:aa:00:01), Dst: 00:00:00 aa:00:03 (00:00:00:aa:00:03)  
 Internet Protocol Version 4, Src: 10.0.0.20, Dst: 10.0.0.21  
 Internet Control Message Protocol

0000 00 00 00 aa 00 03 00 00 00 aa 00 01 08 00 45 00 .....E  
 0010 00 54 37 89 40 00 40 01 ee f7 0a 00 00 14 0a 00 ..T7@.@  
 0020 00 15 08 00 5d 73 00 1e 00 01 53 bc 1e 63 00 00 .....]s..S.c.  
 0030 00 00 69 7b 00 00 00 00 00 00 10 11 12 13 14 15 ..i{.....  
 0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 .....!#\$%  
 0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 &'()\*+,-./012345

wireshark\_veth4.1.5a\_20220912045533\_AnM6p.pcapng Packets: 45 · Displayed: 10 (22.2%) Profile: Default

## Interface 2: (node3)

The image shows a Wireshark capture window titled "veth4.2.5a". The interface has a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help) and a toolbar with various icons. The filter bar shows "ip || arp". The packet list displays four packets:

No.	Time	Source	Destination	Protocol	Length	Info
11	119.177725968	00:00:00 aa:00:01	Broadcast	ARP	42	Who has 10.0.0.21? Tell 10.0.0.20
12	119.177851221	00:00:00 aa:00:03	00:00:00 aa:00:01	ARP	42	10.0.0.21 is at 00:00:00:aa:00:03
13	119.177884493	10.0.0.20	10.0.0.21	ICMP	98	Echo (ping) request id=0x001e, seq=1/
14	119.178127481	10.0.0.21	10.0.0.20	ICMP	98	Echo (ping) reply id=0x001e, seq=1/

The packet details pane shows the selected packet (Frame 11) with the following information:

- Frame 11: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
- Ethernet II, Src: 00:00:00 aa:00:01 (00:00:00:aa:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
- Address Resolution Protocol (request)

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0000 ff ff ff ff ff ff 00 00 00 aa 00 01 08 06 00 01
0010 08 00 06 04 00 01 00 00 00 aa 00 01 0a 00 00 14
0020 00 00 00 00 00 00 0a 00 00 15
```

The status bar at the bottom indicates "wireshark\_veth4.2.5a\_20220912045553\_Q8cUSZ.pcapng", "Packets: 36 · Displayed: 4 (11.1%)", and "Profile: Default".

## Conclusion

Overall I believe we found and were able to complete the investigation with satisfaction however there are some things I would do differently next time such as adding a predefined time interval entry removal for unused cache in the arp table. This would make the overall program more responsible by allowing for monitoring of the ARP data in the table with added removal capabilities.

For the learning switch I was overall satisfied with my completion but felt I should have been able to implement a proper timing function to remove kernels from the table. However, next time I would not only ensure that functioning is successful I would also implement a monitoring method which is responsible for recording the number of entries in the table and ensuring it does not exceed the pre-defined values. This method will be able to prioritize entries and remove unused entries even when the expiration has not finished.

## References

Add/remove/modify ARP tables, Code Project,

<http://www.codeproject.com/Articles/22483/Edit-Add-Remove-Modify-ARP-Tables>

Get the interface Name/Data, MicroHowto,

[http://www.microhowto.info/howto/get\\_the\\_mac\\_address\\_of\\_an\\_ethernet\\_interface\\_in\\_c\\_using\\_siocgifhwaddr.html](http://www.microhowto.info/howto/get_the_mac_address_of_an_ethernet_interface_in_c_using_siocgifhwaddr.html)

Fix Segmentation Fault in C++, DelftStack

<https://www.delftstack.com/howto/cpp/cpp-fix-segmentation-fault/>

## Appendices

### 1. Appendix A - Software

Windows Linux Subsystem, WSL & Ubuntu

<https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-11-with-gui-support#1-overview>

WinSCP 5.21.2, Windows 64bit

<https://winscp.net/eng/download.php>

PuTTY 0.77, Windows 64bit

<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

VirtualBox VM 6.1.38, Windows 64bit

<https://www.virtualbox.org/wiki/Downloads>

### 2. Appendix B - Diagram

Finite State Machine, Diagram

<https://madebyevan.com/fsm/>

### 3. Appendix C - Source Code Part 1

arp\_functions.c

arp\_main.c

my\_arp.c

#### Source Code Part 2

lswitch\_functions.c

lswitch\_main.c

My\_lswitch.c

### 4. CPlusPlus C++ Library

Malloc, strlen, memcpy

<https://cplusplus.com/reference/cstdlib/malloc/>

<https://cplusplus.com/reference/cstring/strlen/>

<https://cplusplus.com/reference/cstring/memcpy/>



## **Acknowledgements**

- Tutor, Zach Kingsford, with coding and development help
- Professors, Winston Seah and Alvin Valera, for development and general networking knowledge