

# Traffic Capture Lab

CYBR3010

Professor: Sam El'Awour

Gregory Stephens

October 14, 2025

<b>Introduction.....</b>	<b>2</b>
<b>Network Diagram (Visio).....</b>	<b>3</b>
<b>3. SPAN Configuration and Verification.....</b>	<b>4</b>
<b>4. Connectivity Verification.....</b>	<b>5</b>
<b>5. ICMP (Ping) Capture.....</b>	<b>5</b>
<b>6. TCP Three-Way Handshake.....</b>	<b>7</b>
<b>7. TCP Four-Way Teardown.....</b>	<b>9</b>
<b>8. DNS (UDP) Traffic.....</b>	<b>10</b>
<b>9. HTTP Burst (Traffic Generation).....</b>	<b>12</b>
<b>10. TTL and MAC Address Analysis.....</b>	<b>13</b>
<b>11. Packet Count Verification.....</b>	<b>15</b>
<b>12. Conclusion.....</b>	<b>16</b>
<b>13. Submission Checklist.....</b>	<b>17</b>

# Introduction

This lab demonstrates the placement and use of a protocol analyzer to capture and analyze switched network traffic at multiple points in a small, VLAN-segmented lab network. The environment is hosted in VMware and consists of a single managed switch (**SW01**), a firewall gateway (**FW01**) that performs inter-VLAN routing and access control, and three virtual machines: **Kali** (**VLAN10, 192.168.10.50**), **Client20** (**VLAN20, 192.168.20.50**), and **Client30** (**VLAN30, 192.168.30.50**). **Client30** is configured as the **SPAN (mirror)** destination so it can passively capture all traffic on the trunk between **SW01** and **FW01**; **Client20** and **Kali** are used as the active source and destination to generate and observe traffic.

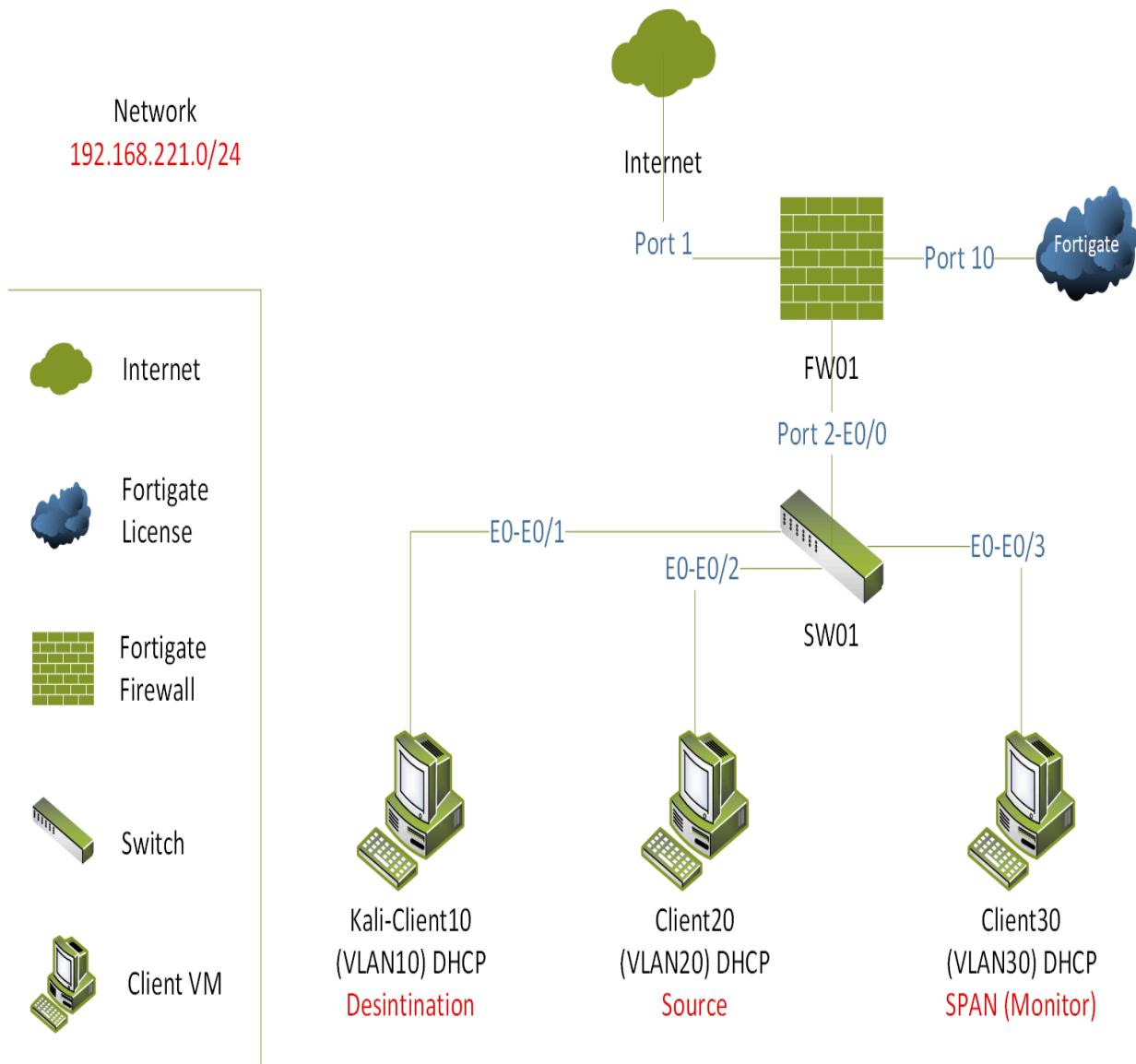
The primary objective is to capture the same flows simultaneously at the source, on the mirrored trunk (**SPAN**), and at the destination to demonstrate switched-network behavior, packet visibility differences, and protocol interactions. Wireshark was used on all three VMs to record **ICMP (ping)**, **TCP (HTTP on a temporary Python server)**, and **UDP (DNS)** traffic. Troubleshooting steps were performed to resolve service binding and firewall policy issues so that **TCP** sessions could be established and examined.

This report includes a network diagram, configuration evidence for SPAN, step-by-step capture points, packet analysis of the **TCP three-way handshake and TCP four-way teardown**, a **TTL/MAC** comparison across capture points, and traffic generation logs (**DNS and HTTP bursts**). All captures are saved and named to map directly to the capture points and traffic types; these files are provided with the submission to allow independent verification and further analysis.

# Network Diagram (Visio)

## Network Topology

Gregory Stephens, Lab 3 Network Design



### 3. SPAN Configuration and Verification

How this screenshot was captured:

On SW01, the following commands were used:

```
monitor session 1 source interface e0/0 both  
monitor session 1 destination interface e0/3  
show monitor session 1
```

Screenshots were taken from the SW01 console during configuration and verification.

The screenshot shows the SW01 terminal window. The user has entered configuration mode with 'configure terminal'. They then created a monitor session named 'session 1' with the command 'monitor session 1'. This session is configured to have 'source interface ethernet0/2 both' and 'destination interface ethernet0/3'. Finally, they checked the configuration with 'show monitor session 1', which displayed the session details: Type Local Session, Source Ports Both (Et0/2), Destination Ports Et0/3, and Encapsulation Native. The terminal window also shows system logs at the top and resource usage at the bottom.

```
>_ SW01  
*Oct 10 23:10:51.298: %AMDP2_FE-6-EXCESSCOLL: Ethernet0/2 TDR=0, TRC=0  
*Oct 10 23:11:36.678: %AMDP2_FE-6-EXCESSCOLL: Ethernet0/2 TDR=0, TRC=0  
*Oct 10 23:12:07.318: %AMDP2_FE-6-EXCESSCOLL: Ethernet0/0 TDR=0, TRC=0  
SW01>  
SW01>configure terminal  
^  
% Invalid input detected at '^' marker.  
SW01>enable  
SW01#configure terminal  
Enter configuration commands, one per line. End with CNTL/Z.  
SW01(config)#monitor session 1 source interface ethernet0/2 both  
SW01(config)#monitor session 1 destination interface ethernet0/3  
SW01(config)#exit  
SW01#show  
*Oct 10 23:58:29.811: %SYS-5-CONFIG_I: Configured from console by console  
SW01#show monitor session 1  
Session 1  
-----  
Type : Local Session  
Source Ports :  
    Both : Et0/2  
Destination Ports : Et0/3  
    Encapsulation : Native  
SW01#
```

Fig02\_SPAN\_Config

The screenshot shows the SW01 terminal window. The user has entered configuration mode ('configure terminal') and deleted the previously created monitor session 'session 1' with 'no monitor session 1'. They then re-created the session with 'monitor session 1 source interface e0/0 both destination interface e0/3'. After saving the configuration ('wr'), they checked the session again with 'show monitor session 1', which showed the session details: Type Local Session, Source Ports Both (Et0/0), Destination Ports Et0/3, and Encapsulation Native. The terminal window also shows system logs at the top and resource usage at the bottom.

```
KALI-CL..VLAN10) CLIENT2..VLAN20) >_ SW01  
SW01(config)#monitor session 1  
*Oct 12 16:50:27.337: %AMDP2_FE-6-EXCESSCOLL: Ethernet0/2 TDR=0, TRC=0  
SW01(config)#monitor session 1  
% Incomplete command.  
SW01(config)#no monitor session 1  
SW01(config)#monitor session 1 source interface e0/0 both  
SW01(config)#monitor  
*Oct 12 16:53:04.436: %AMDP2_FE-6-EXCESSCOLL: Ethernet0/3 TDR=0, TRC=0  
SW01(config)#monitor session 1 destination interface e0/3  
SW01(config)#end  
SW01#wr  
*Oct 12 16:53:22.813: %SYS-5-CONFIG_I: Configured from console by console  
SW01#write memory  
Building configuration...  
[OK]  
SW01#show monitor session 1  
Session 1  
-----  
Type : Local Session  
Source Ports :  
    Both : Et0/0  
Destination Ports : Et0/3  
    Encapsulation : Native  
SW01#
```

Fig03\_SPAN\_Verify

## 4. Connectivity Verification

How this screenshot was captured:

Before beginning traffic analysis, Test-NetConnection was run on Client20 to confirm network reachability.

Kali's HTTP server was started to verify TCP access through the firewall.

Commands used:

- Kali: **sudo python3 -m http.server 8080 --bind 192.168.10.50**
- Client20: **Test-NetConnection 192.168.10.50 -Port 8080**

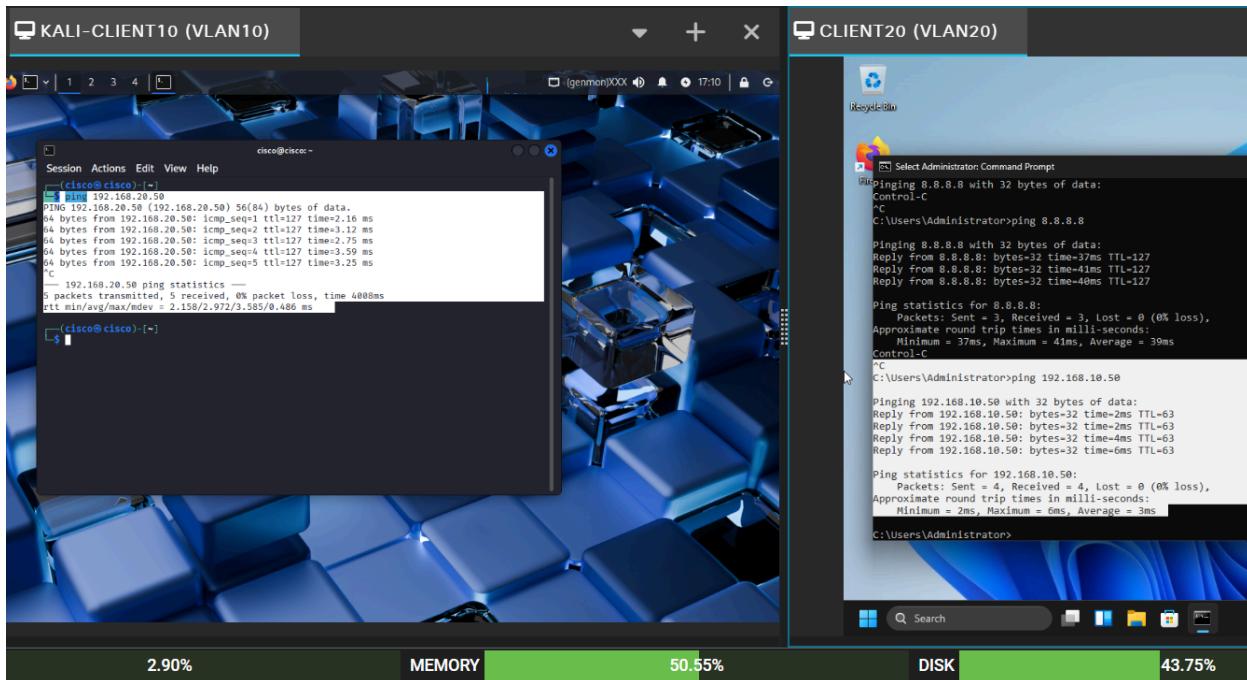


Fig01\_Connectivity\_Test

## 5. ICMP (Ping) Capture

How this screenshot was captured:

Wireshark was started on all three VMs. From Client20:

**ping 192.168.10.50**

Display filter applied:

**icmp && ip.addr == 192.168.10.50 && ip.addr == 192.168.20.50**

Screenshots were taken after several pings to confirm traffic flow.

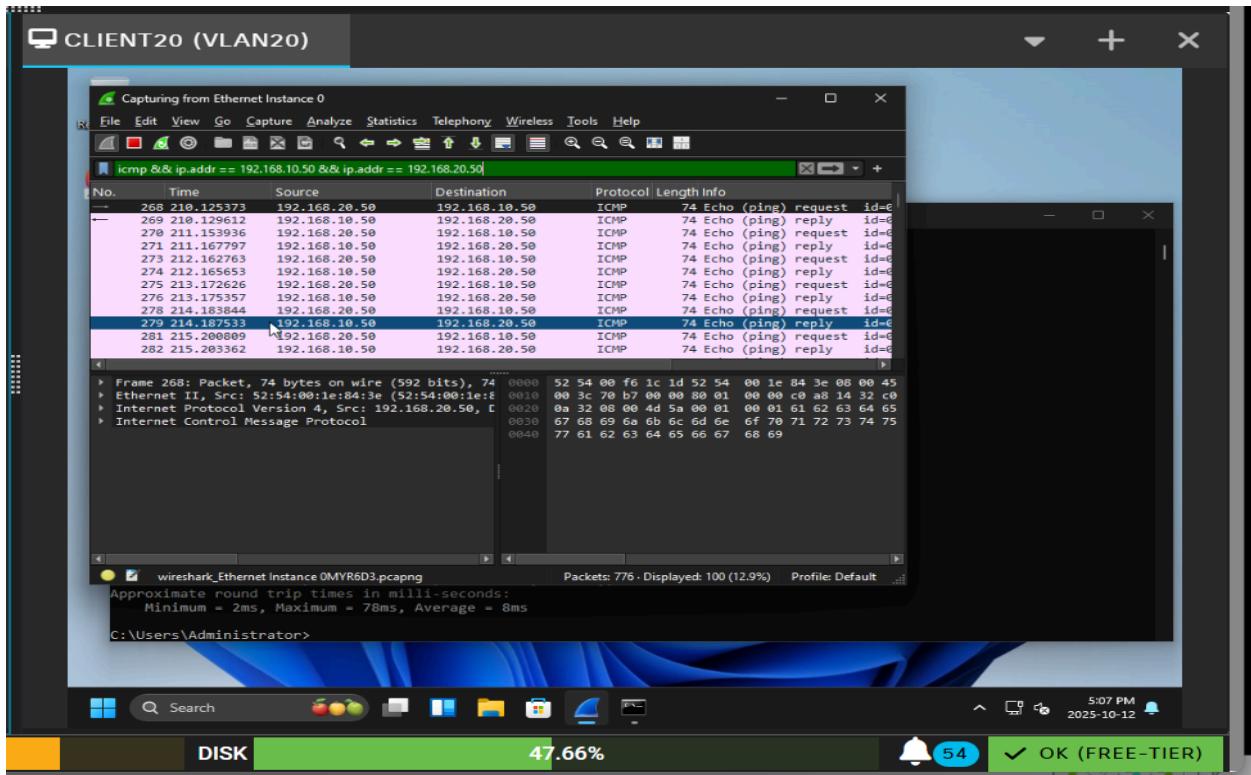


Fig04\_ICMP\_Source

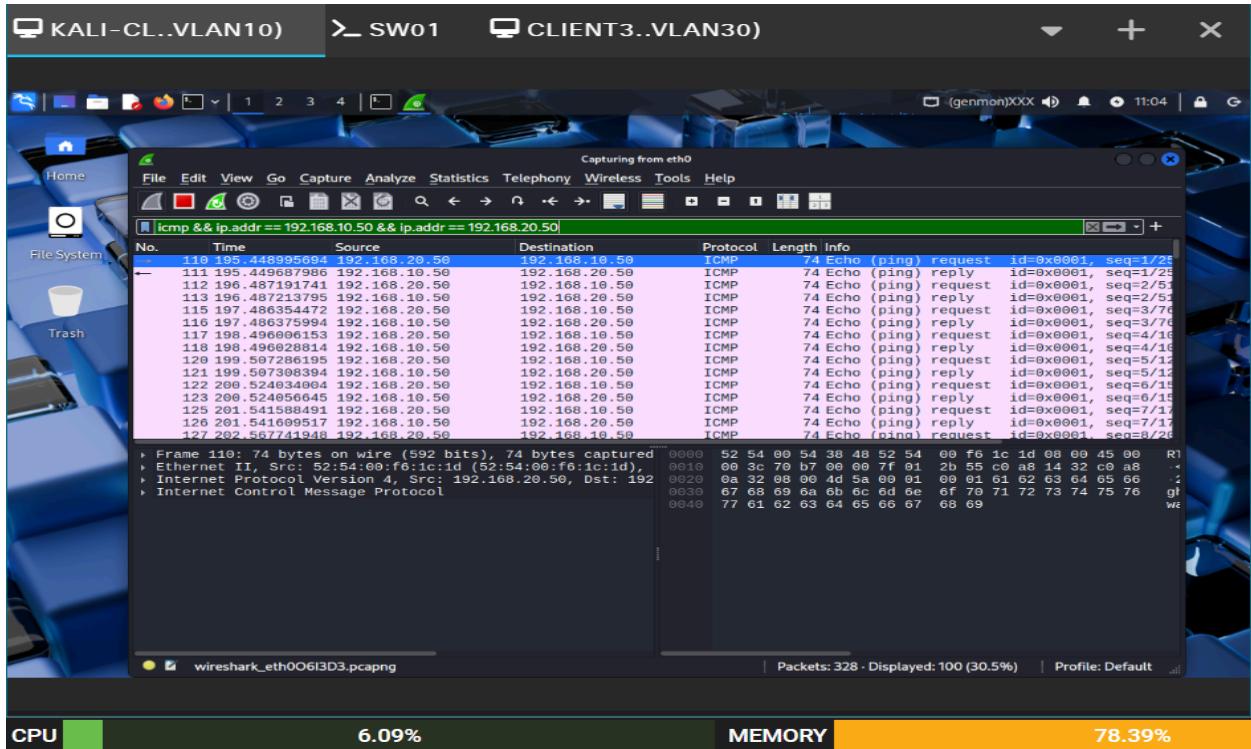
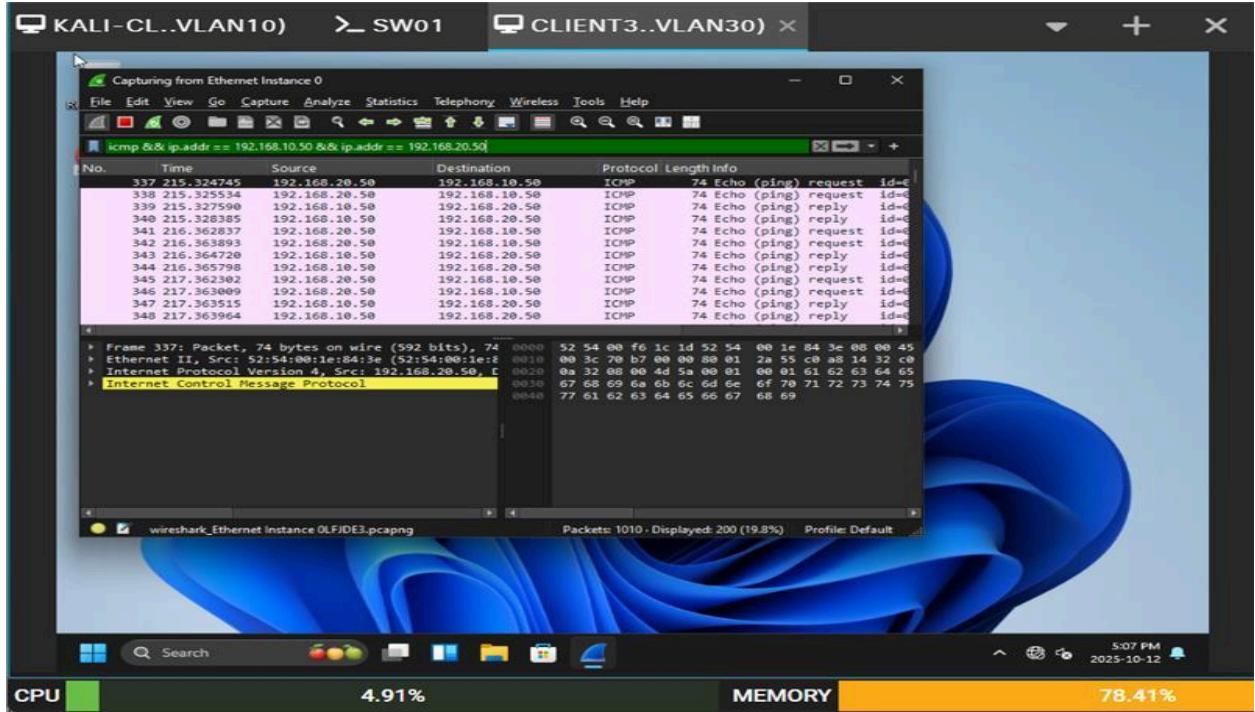


Fig05\_ICMP\_Destination



*Fig06\_ICMP\_SPAN*

#### Explanation:

ICMP packets verify that VLAN routing and SPAN mirroring work correctly. Echo requests and replies were visible on all capture points.

#### Capture Files:

*CAP-Source\_Client20\_ICMP.pcapng*  
*CAP-SPAN\_Client30\_ICMP.pcapng*  
*CAP-Destination\_Kali\_ICMP.pcapng*

## 6. TCP Three-Way Handshake

#### How this screenshot was captured:

After verifying TCP success, Wireshark captured HTTP traffic using:

**tcp && ip.addr==192.168.10.50 && ip.addr==192.168.20.50 && tcp.port==8080**

Packets were identified as:

- **SYN → Client20 to Kali**
- **SYN/ACK → Kali to Client20**
- **ACK → Client20 to Kali**

The first SYN packet was expanded in the “Transmission Control Protocol” section to show flags and TCP options.

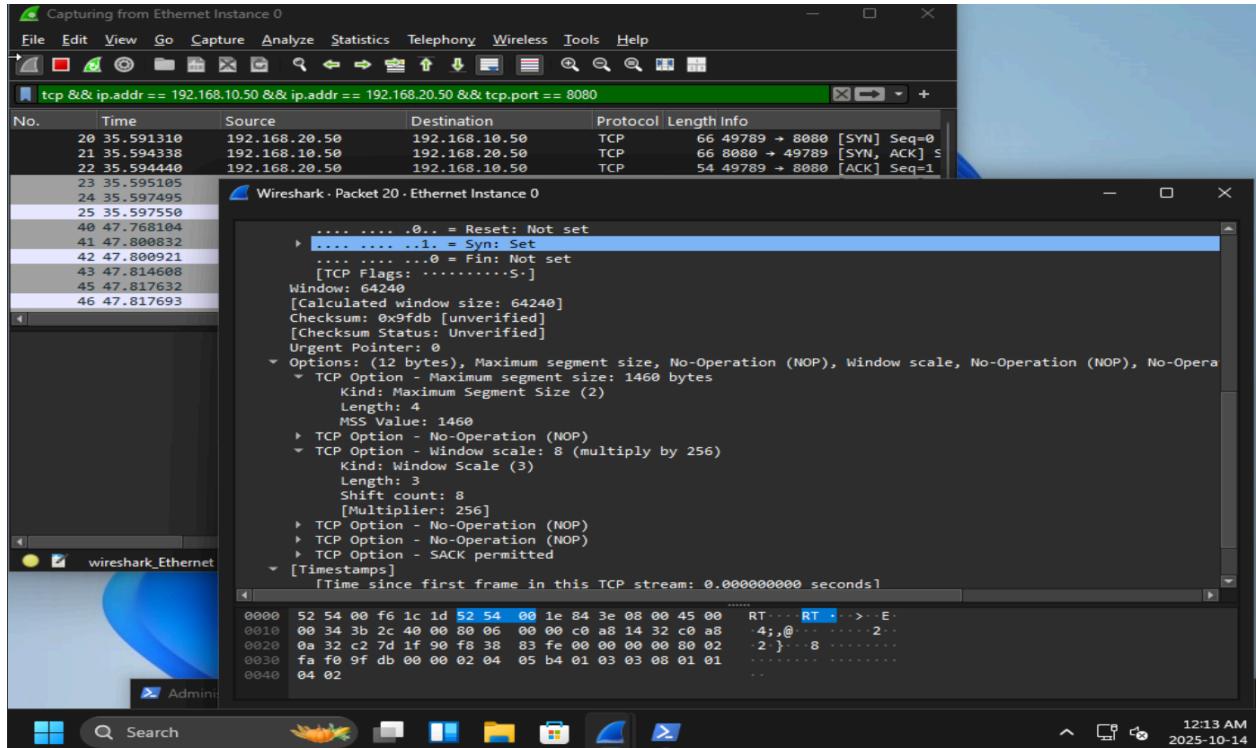


Fig07\_TCP\_3WH\_Source

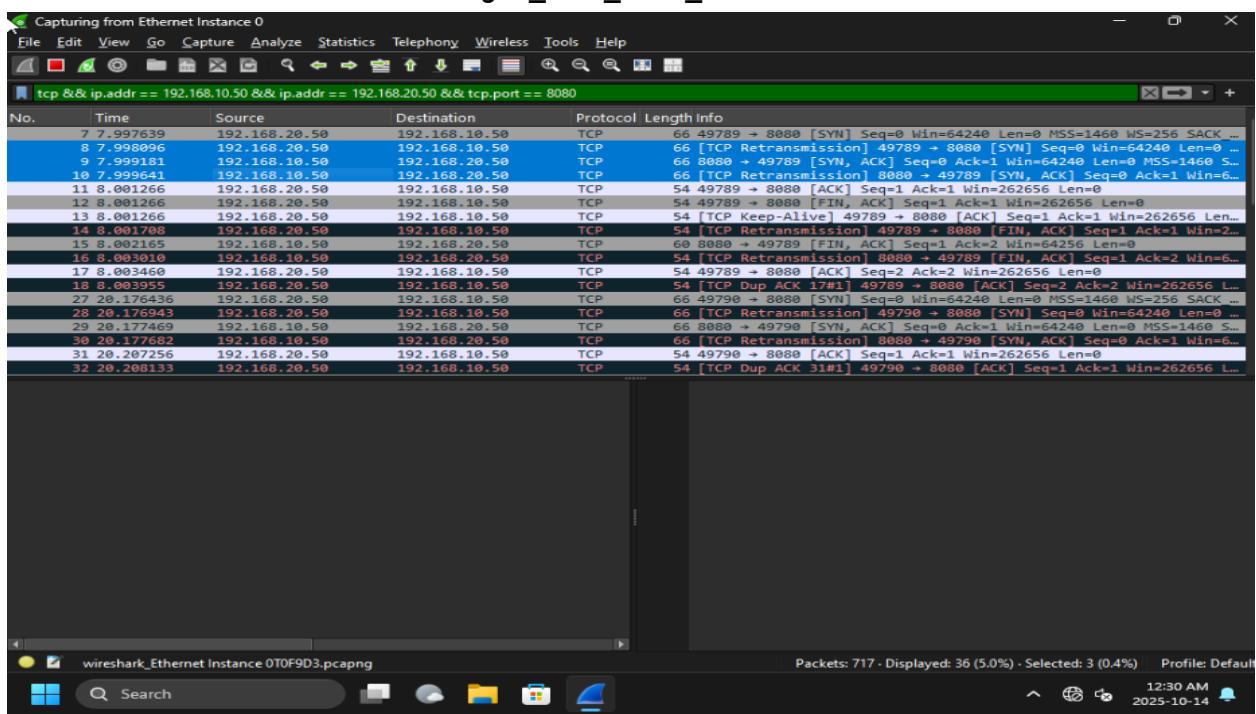
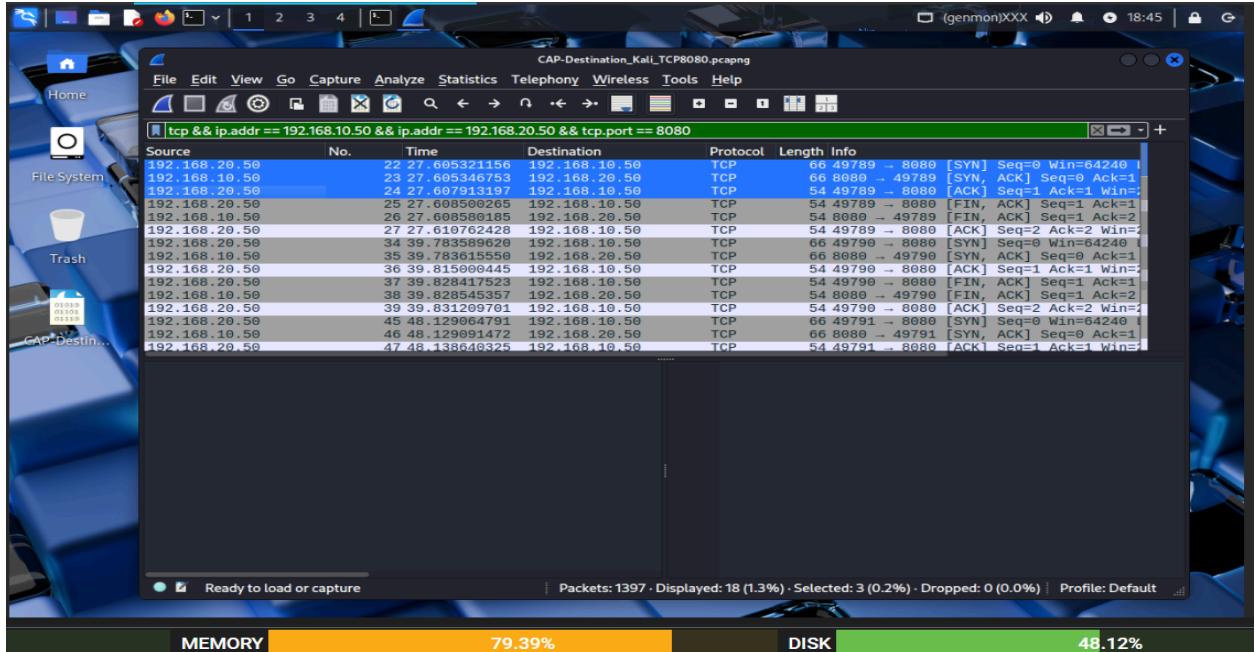


Fig08\_TCP\_3WH\_SPAN



*Fig09\_TCP\_3WH\_Destination*

#### **Explanation:**

The three packets form the TCP handshake that synchronizes sequence numbers. TCP Options observed: MSS 1460, Window Scale 8, SACK Permitted, and Timestamps.

#### **Capture Files:**

*CAP-Source\_Client20\_TCP8080.pcapng*  
*CAP-SPAN\_Client30\_TCP8080.pcapng*  
*CAP-Destination\_Kali\_TCP8080.pcapng*

## 7. TCP Four-Way Teardown

#### **How this screenshot was captured:**

After completing the connection, the client closed the session (browser or PowerShell stopped). The final four packets in the TCP stream were captured, showing:

**FIN/ACK → ACK → FIN/ACK → ACK.**

#### **Explanation:**

This teardown confirms that each side closed its connection gracefully. It demonstrates proper TCP session termination between VLANs through FW01.

### Capture Files:

*CAP-Source\_Client20\_TCP8080.pcapng  
CAP-SPAN\_Client30\_TCP8080.pcapng  
CAP-Destination\_Kali\_TCP8080.pcapng*

## 8. DNS (UDP) Traffic

*How this screenshot was captured:*

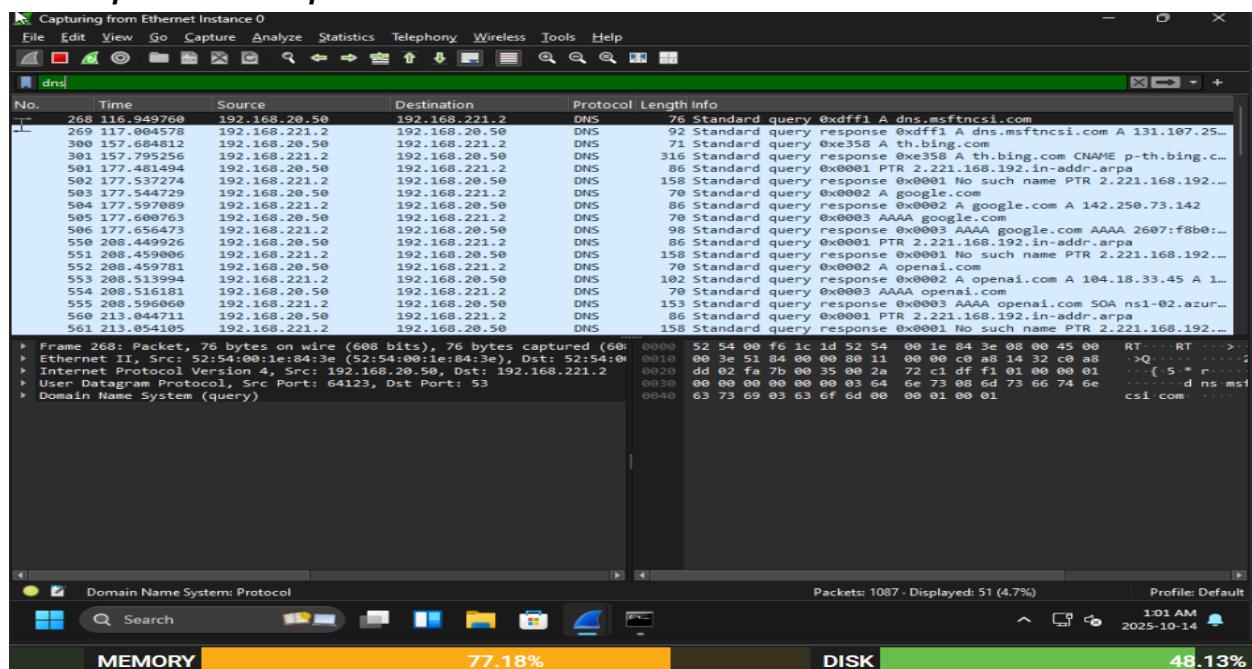
From Client20: (*Powershell*)

*nslookup nait.ca  
nslookup google.com  
nslookup yahoo.com*

Wireshark filter:

*dns*

*DNS requests and responses were visible on all three VMs.*



*Fig13\_DNS\_Source*

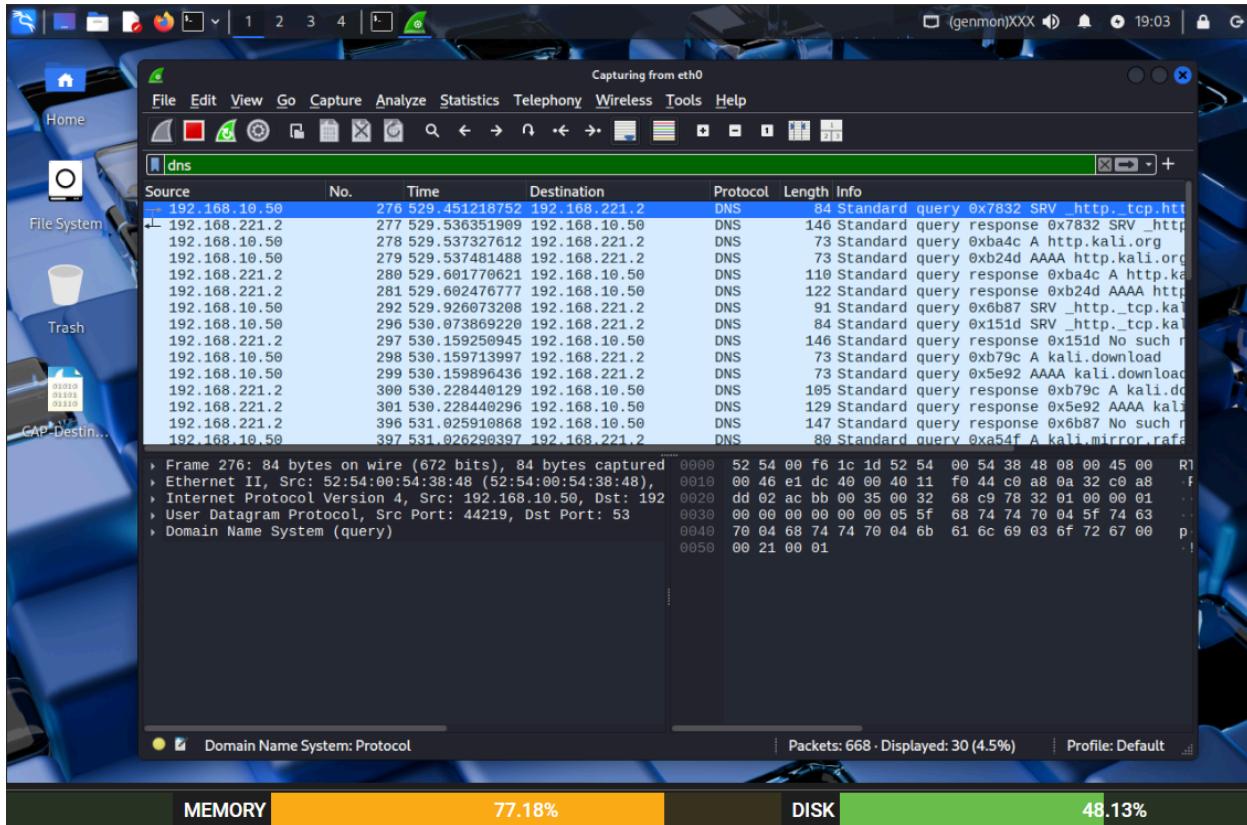


Fig14\_DNS\_Destination

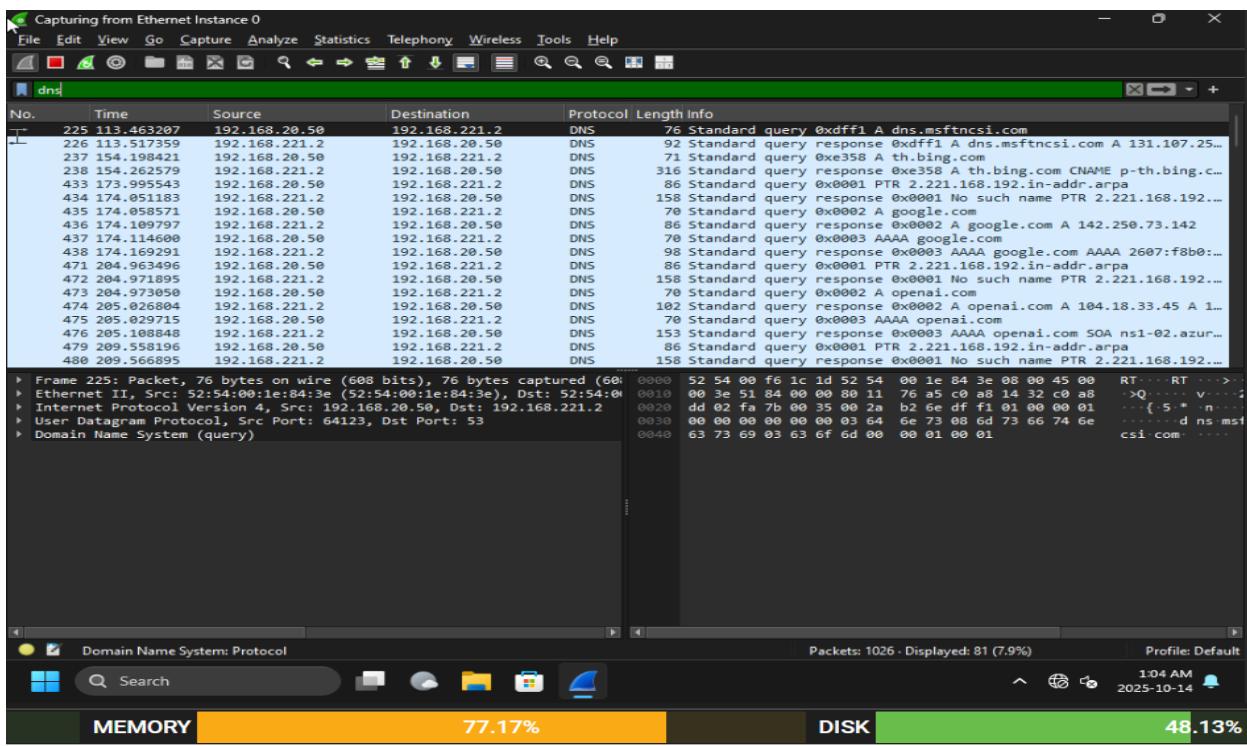


Fig15\_DNS\_SPAN

## Explanation:

Each query uses **UDP** port 53. Queries and responses confirm **UDP** traffic flow across VLANs and **SPAN** mirroring.

## Capture Files:

*CAP-Source\_Client20\_DNS.pcapng*  
*CAP-SPAN\_Client30\_DNS.pcapng*  
*CAP-Destination\_Kali\_DNS.pcapng*

## 9. HTTP Burst (Traffic Generation)

### How this screenshot was captured:

From Client20: (*Powershell*)

```
for ($i=0; $i -lt 20; $i++) { Invoke-WebRequest -Uri http://192.168.10.50:8080 -UseBasicParsing }
```

Wireshark filter:

**tcp.port==8080**

Screenshot captured showing multiple HTTP GET requests in sequence.

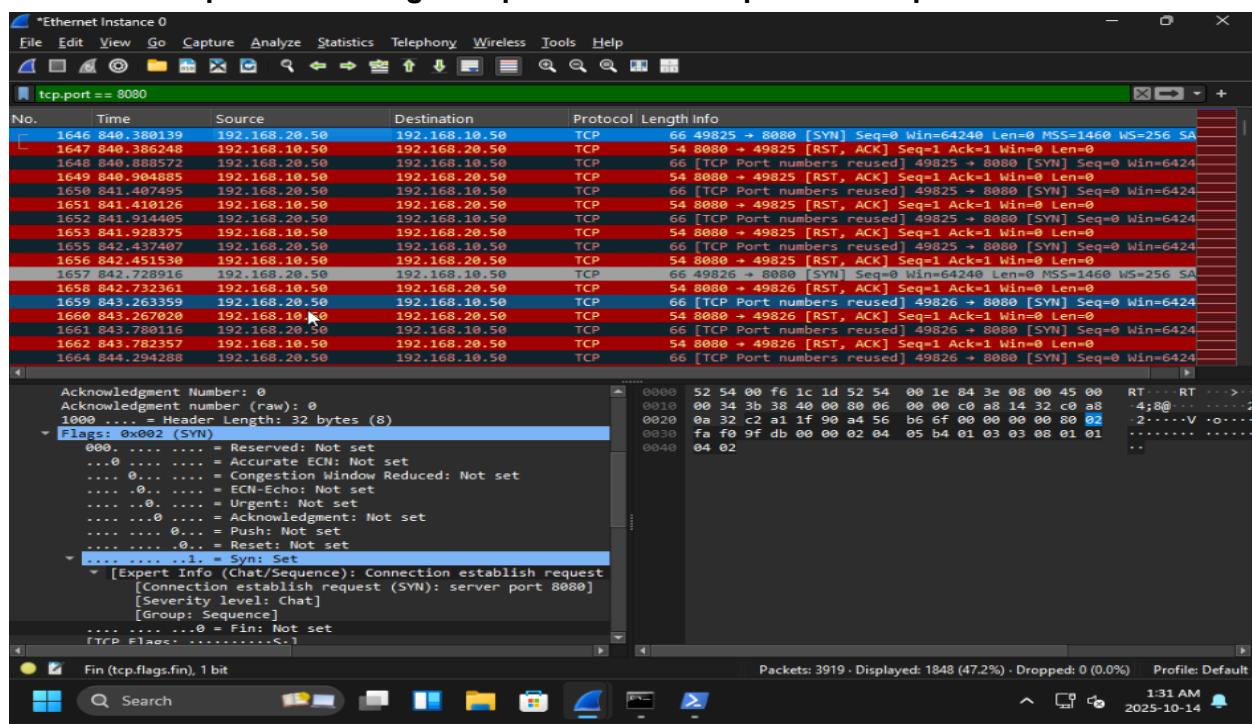


Fig16\_HTTP\_Burst

## Explanation:

Multiple concurrent **TCP** sessions simulate real-world traffic and increase the total packet count to meet lab requirements.

## Capture Files:

*CAP-Source\_Client20\_HTTPBurst.pcapng*  
*CAP-SPAN\_Client30\_HTTPBurst.pcapng*  
*CAP-Destination\_Kali\_HTTPBurst.pcapng*

## 10. TTL and MAC Address Analysis

### How this screenshot was captured:

Display filter applied:

**icmp && ip.src==192.168.20.50 && ip.dst==192.168.10.50**

Expanded Ethernet II and IPv4 Header in the same packet across all captures to display TTL, source MAC, and destination MAC.

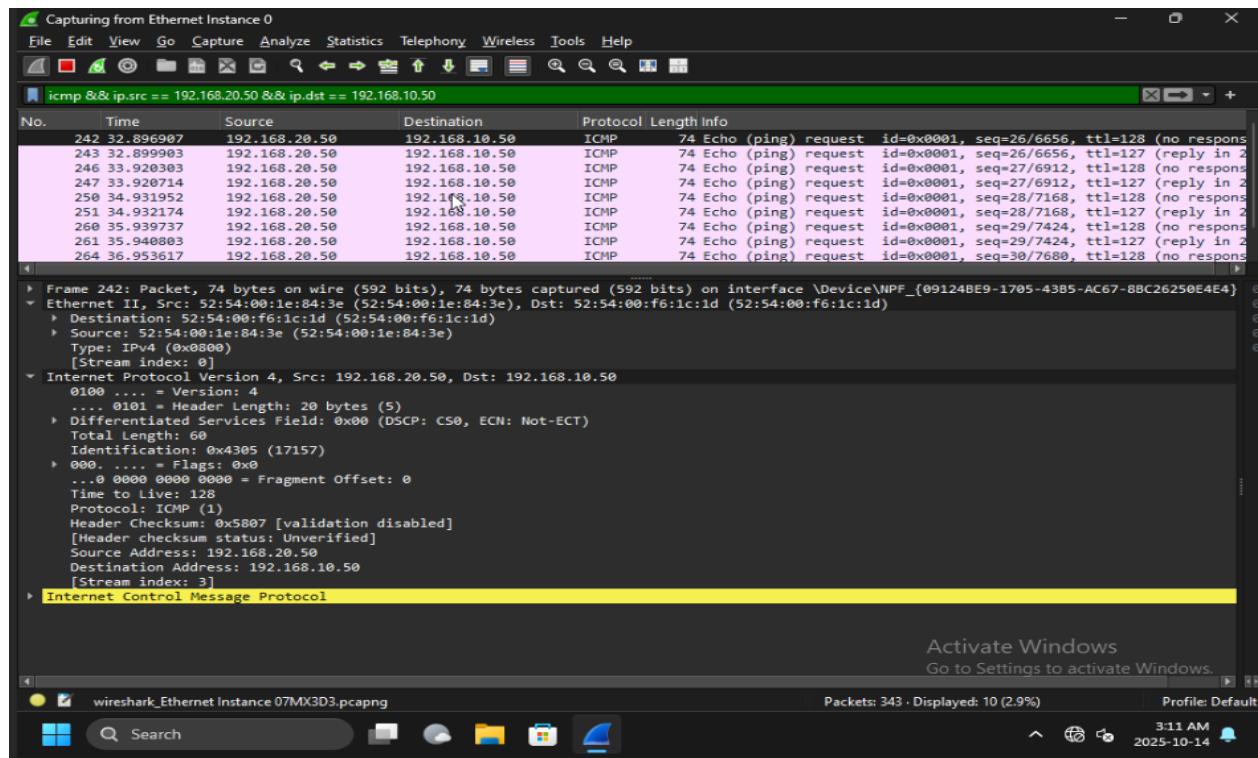


Fig17\_TTL\_Source

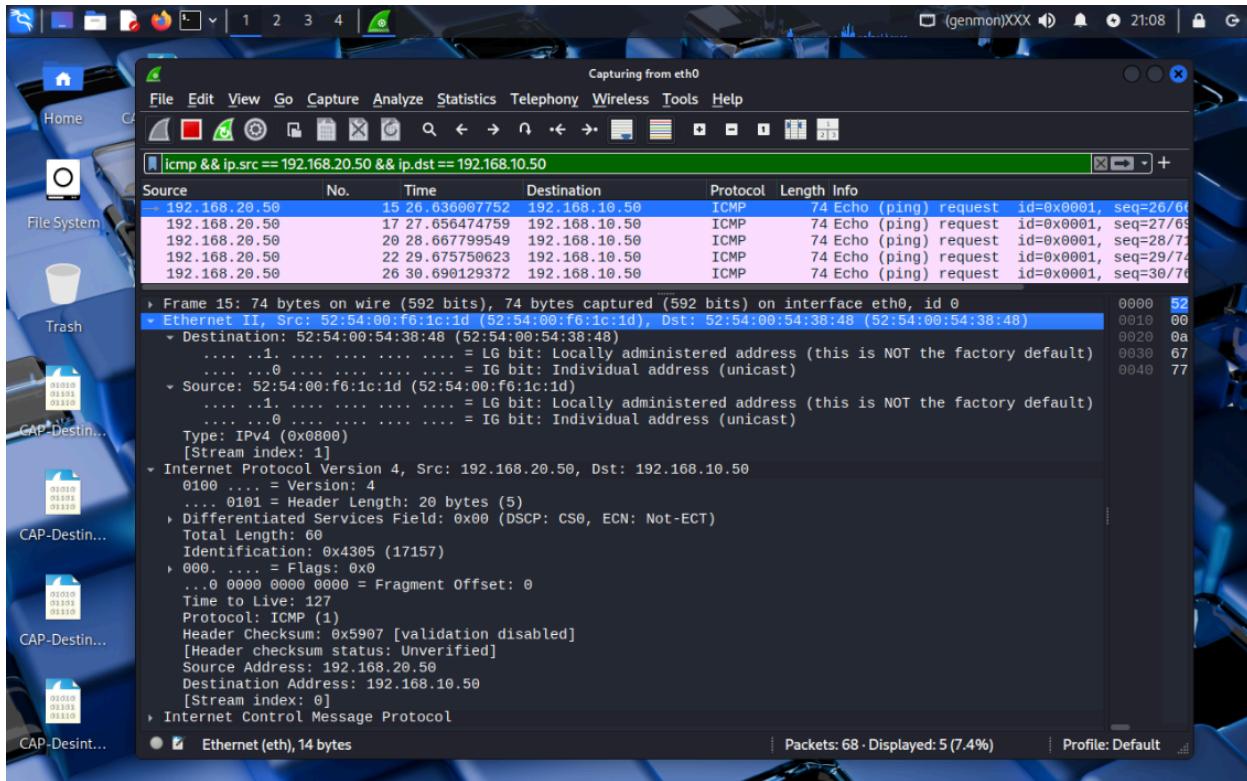


Fig18\_TTL\_Destination

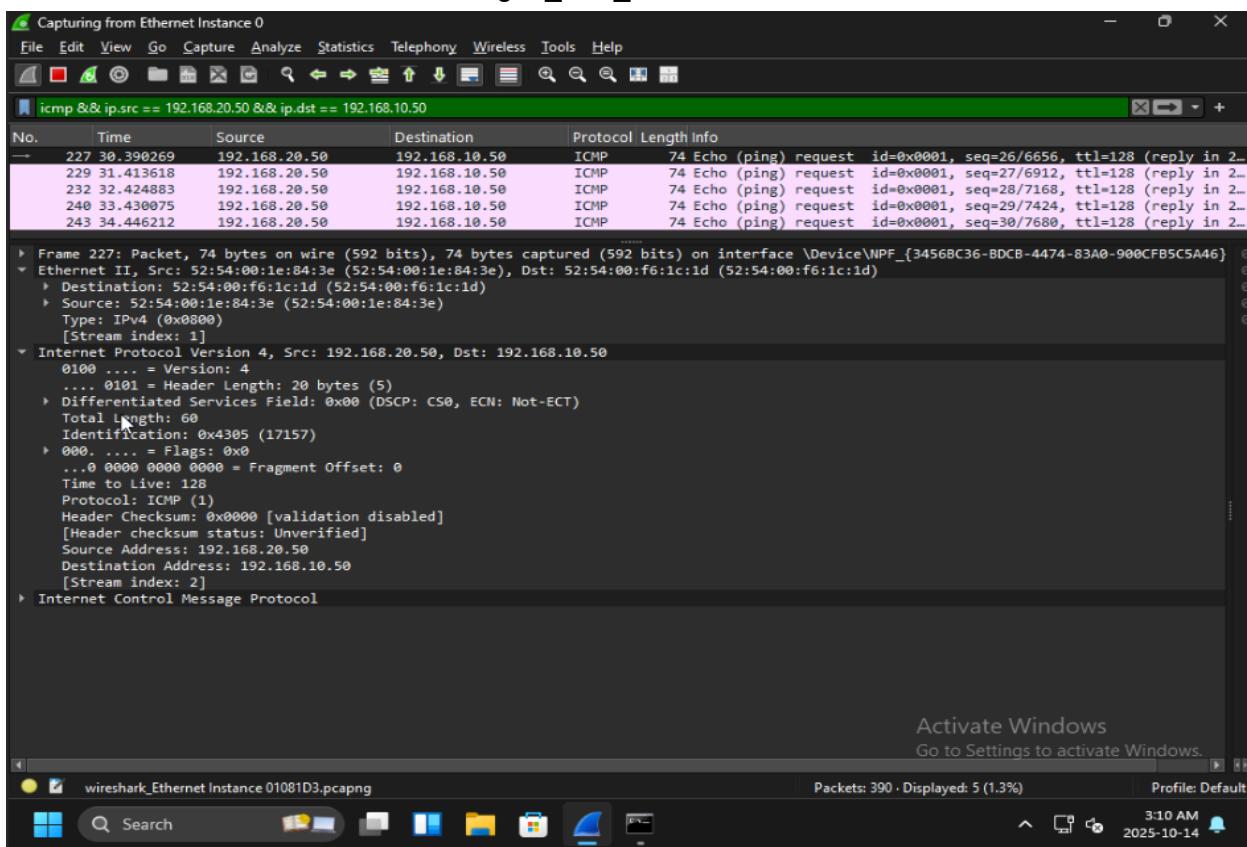


Fig19\_TTL\_SPAN

### **Explanation:**

TTL decreases by one at each Layer 3 hop, and MAC addresses change at each Layer 2 segment since routers rewrite Ethernet headers before forwarding.

Capture Point	Filter Used	TTL	Source MAC	Destination MAC
Source (Client20)	same	[TTL value]	[MAC]	[MAC]
Destination (Kali)	same	[TTL value]	[MAC]	[MAC]
SPAN (Client30)	same	[TTL value]	[MAC]	[MAC]

### **Capture Files:**

*CAP-Source\_Client20\_TTL.pcapng*

*CAP-SPAN\_Client30\_TTL.pcapng*

*CAP-Destination\_Kali\_TTL.pcapng*

## 11. Packet Count Verification

### **How this screenshot was captured:**

In Wireshark:

**Statistics → Capture File Properties**

**Each file shows a total packet count exceeding 3,000.**

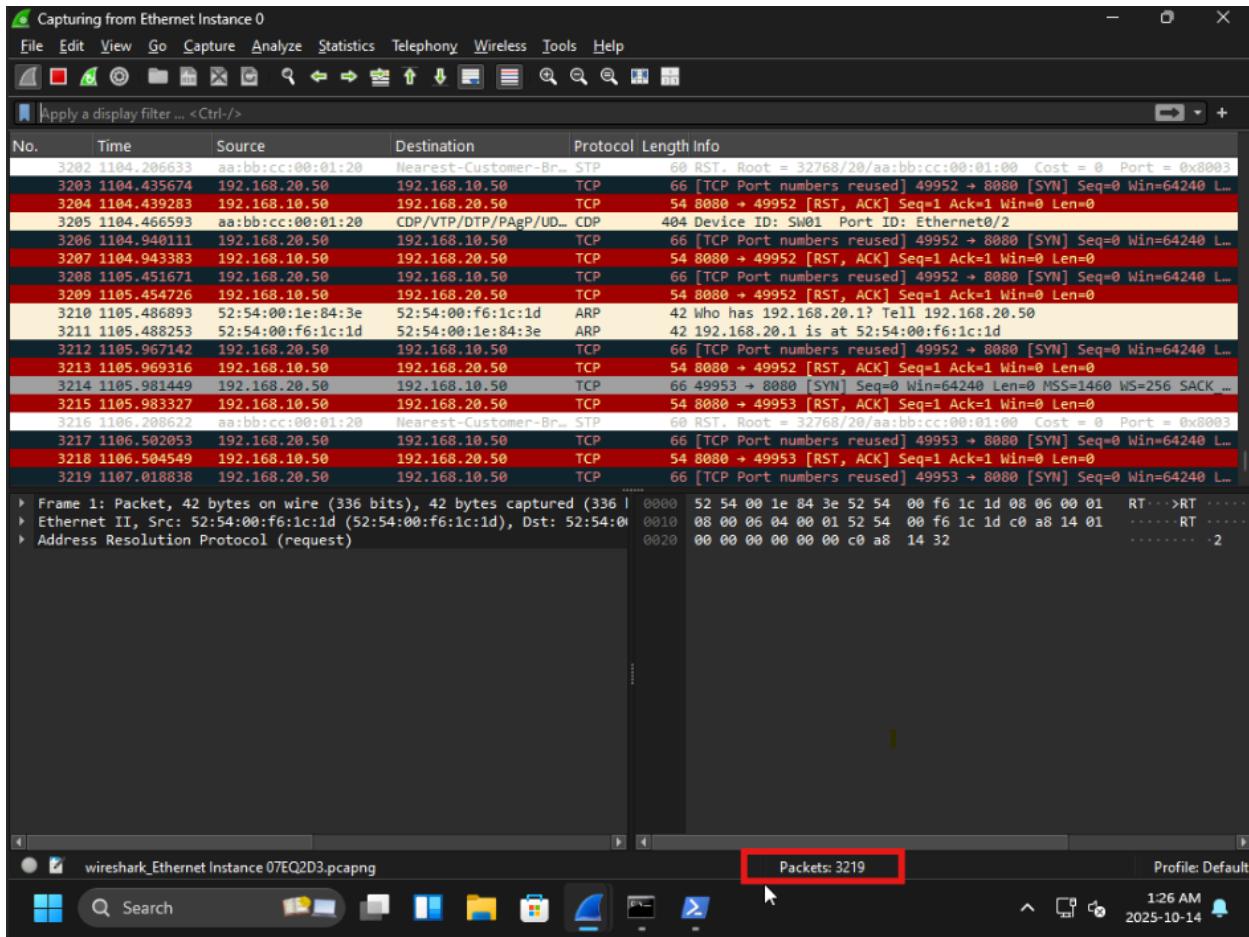


Fig20\_TotalPackets

## 12. Conclusion

The Traffic Capture Lab demonstrated the ability to configure a **SPAN** session, generate and analyze traffic across VLANs, and use Wireshark to interpret **ICMP**, **TCP**, and **UDP** traffic. The **TCP three-way handshake** and **four-way teardown** were successfully identified, **DNS** verified **UDP** communication, and **TTL** analysis confirmed proper Layer 2/3 operation.

## 13. Submission Checklist

### **Submit as:**

CYBR3010\_TrafficCapture\_GregoryStephens.zip

### **Contents:**

- CYBR3010\_TrafficCapture\_Report\_GregoryStephens.pdf
- All figures (Fig00 → Fig18)
- All .pcapng capture files:

*CAP-Source\_Client20\_ICMP.pcapng*

*CAP-SPAN\_Client30\_ICMP.pcapng*

*CAP-Destination\_Kali\_ICMP.pcapng*

*CAP-Source\_Client20\_TCP8080.pcapng*

*CAP-SPAN\_Client30\_TCP8080.pcapng*

*CAP-Destination\_Kali\_TCP8080.pcapng*

*CAP-Source\_Client20\_DNS.pcapng*

*CAP-SPAN\_Client30\_DNS.pcapng*

*CAP-Destination\_Kali\_DNS.pcapng*

*CAP-Source\_Client20\_HTTPBurst.pcapng*

*CAP-SPAN\_Client30\_HTTPBurst.pcapng*

*CAP-Destination\_Kali\_HTTPBurst.pcapng*

*CAP-Source\_Client20\_TTL.pcapng*

*CAP-SPAN\_Client30\_TTL.pcapng*

*CAP-Destination\_Kali\_TTL.pcapng*