CSL6010 Cyber Security
Assignment 04

Name: Sumeet S Patil
Roll No: B22CS052

Objective: The aim of this lab is to analyze network traffic using Wireshark to understand protocol behavior, measure delays, and compare secure and insecure methods of data transmission, including credential exposure and encryption.

Task 01:

Task 1.1: Local FTP Server using terminal

FTP Server was started using the terminal using the command **ftp**

Upon capturing the packets using Wireshark for linux, we could notice a few essential packets as highlighted in Fig 1 for Username. It was noticed that there was no encryption of the data being transmitted and the username **test1** could be easily captured by anyone in the middle eavesdropping and could impersonate us. In the same way, password too (Fig 2) was being sent without encryption thus allowing its interception in the middle completing all the credentials needed for the attacker to impersonate others and get access to the resources illegally.

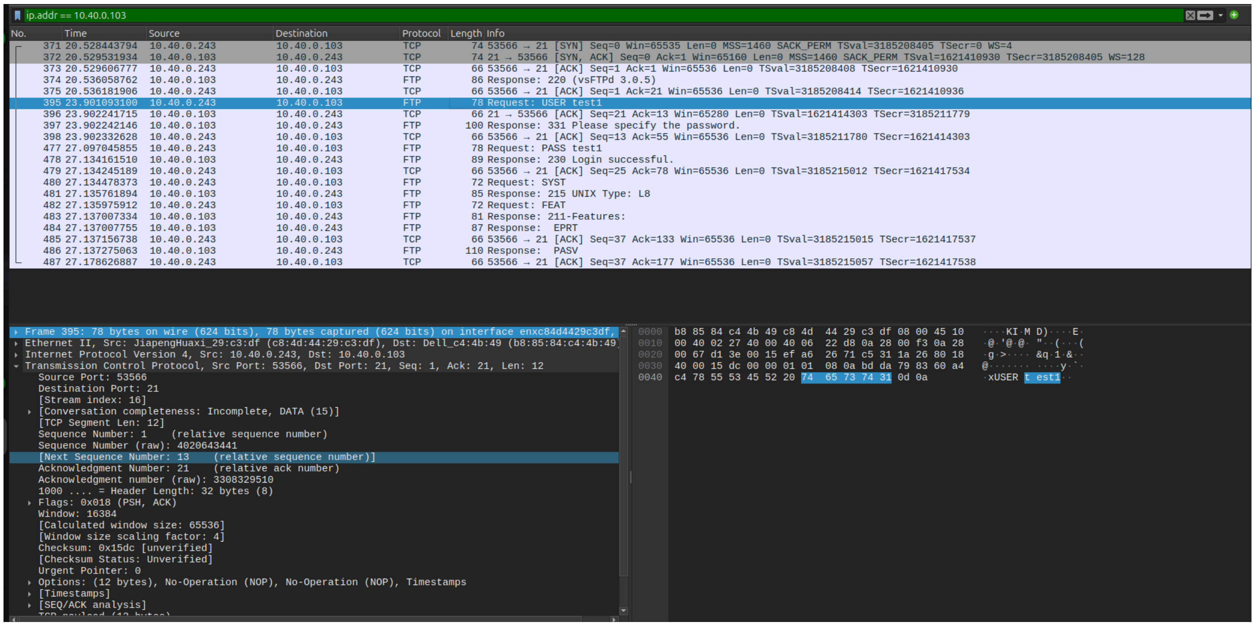

Fig 1: TCP Protocol for FTP Server (Username)

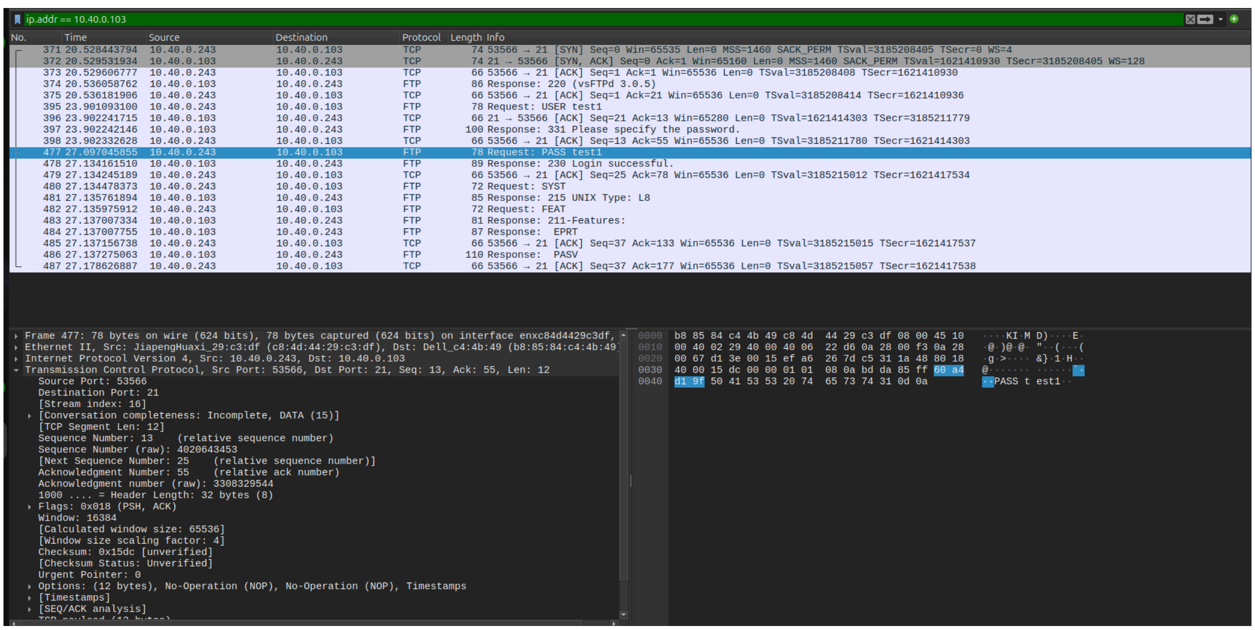

Fig 2: TCP Protocol for FTP Server (Password)

Task 1.2 FTP Server login using http

The browser Google Chrome was used to send a **GET** request to : http://10.40.0.103 which led to the html page to login. Upon entering the credentials, we stopped capturing the packets and started observing.

Upon observation, it was found that the user name and password were url encoded **(Fig 3)**, meaning part of the url of the end point corresponding to login. This is very dangerous, since the eavesdropper needn't look at the captured packet's inside data as well. The endpoint URL is quite enough to get hold of the credentials, which is quite alarming.

Task 1.3 HTTPS POST request for login

A HTTPS POST request was made to the IITJ gateway to login for Internet access and the packets' capture were stopped soon after there was a response for the server. We could notice that the POST request was actually handled by TSLv1.3, which actually maintains security, by encryption. The credentials were to be sent in the packets with labels like "Application Data" as the credentials were entered in the application layer of the network. Upon diving into further details of the packets with application data labels, we noticed that each of their data is encrypted. The encryption key is actually set first, and then the communication begins. **Fig 4** shows one such instance of an application data packet, where the encrypted data is visible. Thus, even after eavesdropping, the attacker can't get the credentials in a straightforward manner. Thus, **computational security** is achieved. In order to counter brute force attacks, I could observe a strategy, not sure whether that is the way it works. I could notice packets periodically with the description **Change of Cipher**, now these would mean probably a change of the key I guess, thus countering the brute force attacks, since after the key is changed, the ciphers also change, leading to waste of the previous information the attacker had about the old key.

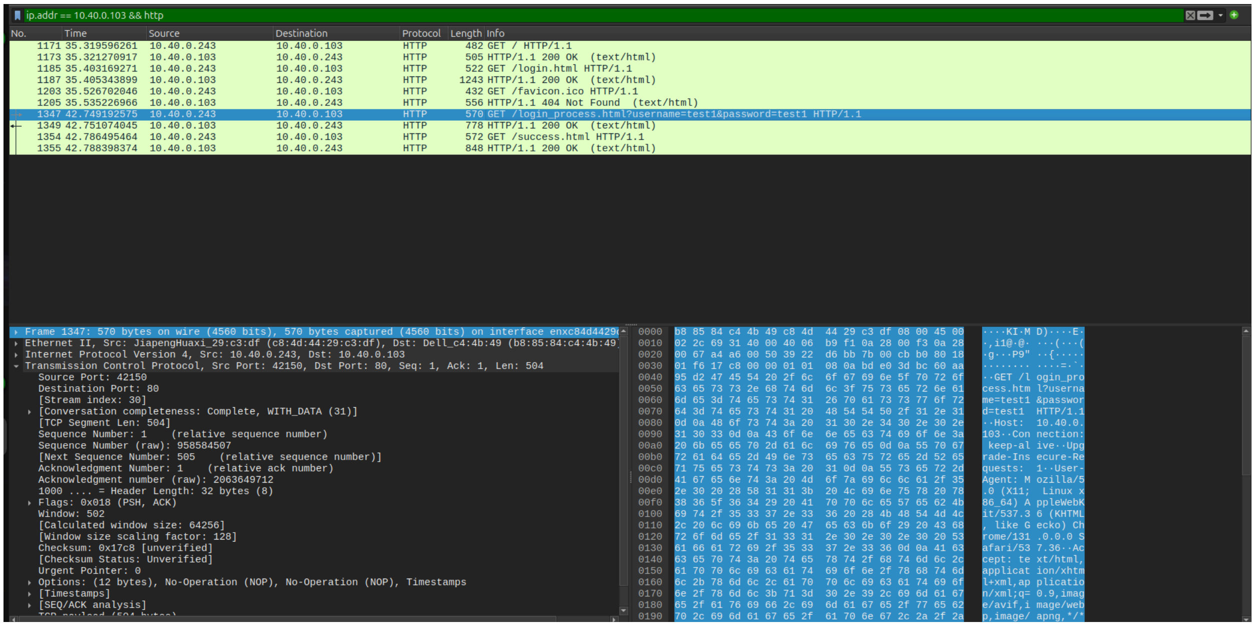

Fig 3: HTTP packets for FTP Server

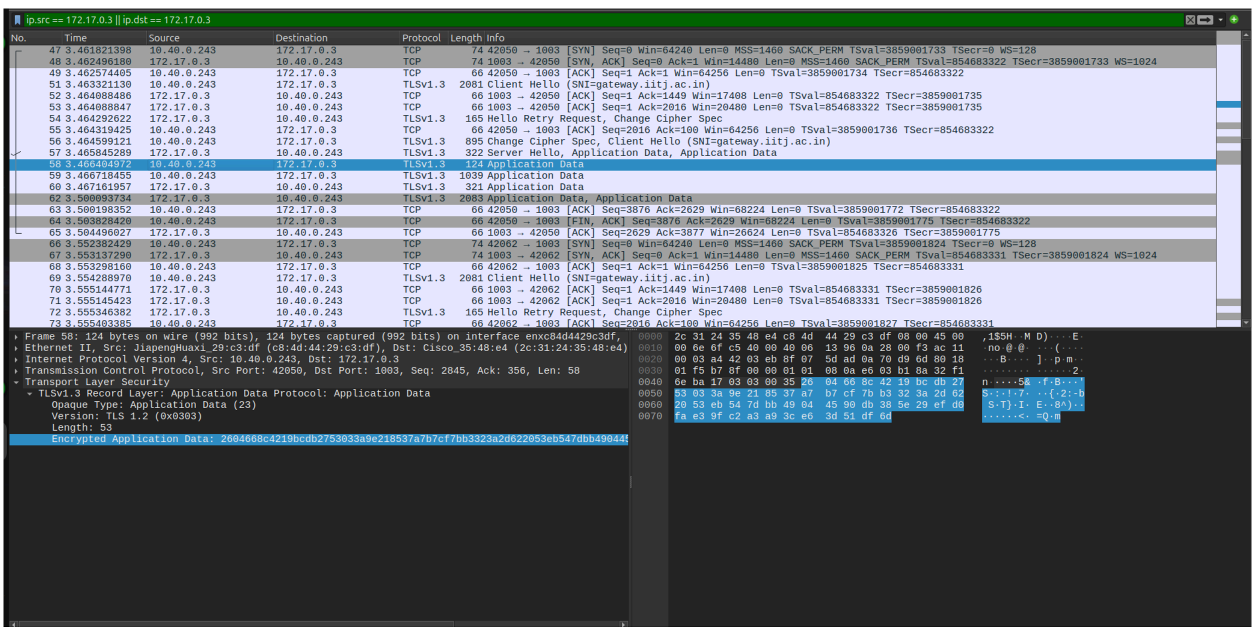

Fig 4: HTTPS packets to IITJ gateway

Task 02:

TCP handshake delay is basically the delay caused due to TCP handshake for establishing the TCP connection between two machines.

Usually TCP handshake delay is equal to the sum of time between syn ,syn ack signals and syn ack , ack signals. Mathematically in Wireshark,
TCP handshake delay = (SYN ACK TS) - (SYN TS) + (ACK TS) - (SYN ACK TS)
Where TS = timestamp

Now, the TCP handshake is a very crucial part of the TCP/IP protocol, since this establishes the TCP connection between two machines correctly.
Impact of TCP handshake delay on network performance:

1. Increase in its delay will slow down the communication between machines, especially for short lived connections like HTTP requests.
2. Websites relying on the multiple TCP connections will have too much slowed down page loading, since resources are fetched using different TCP connections, each beginning with a handshake.

How handshake delays can be analyzed:

1. If there is too much delay in the syn and ack signals, it can be due to congestion on the network, due to packet losses or transmission delays
2. Also, the server might be quite slow, thus having the issue of processing time and in turn the delay.

Methods to analyze application level delays in wireshark:

1. For HTTP, we can set the filter **http.request || http.response**, thus the time between these for a particular source and destination IP, the delays can be known.
2. Same way for DNS queries and responses, filters set to **dns** would help analyze.

Ways to optimize:

1. If websites have a long load time, then we can reduce the handshakes by keeping **persistent TCP connections.**
2. Having **local caching** of DNS data (Domain name,IP) pairs to eliminate dns requests can help boost the communication by reducing latency.
3. Finally, using HTTP/2 or HTTP/3 can help in reducing the application level latency, since they are known for faster set up times.

Task 03:

The pcap downloaded is: **File**

Looking at the capture for a bird eye view, it was clearly visible that it was a UDP attack. To confirm it, I applied the filter “udp” and noticed that out of the total 13804 packets captured, 9657 (70%) were displayed after the filter (Fig 5). Thus, it was confirmed to be a UDP attack. The next task(s) were to determine:

a. How to differentiate between legitimate and attack traffic
b. The attack IP address(es)
c. Analyzing its impact of the services of the server

To differentiate legitimate traffic from attack traffic, I went to **Statistics->Conversations->UDP**

Then sorting by number of packets sent, there was a clear picture who the attacker was as seen in **Fig 6.** The IP **192.168.0.2** sent 7477 UDP packets to an IP **192.168.0.1**, thus being the sole attackers, since the other sources hardly had above 50 packets. Thus, by applying this method, we could achieve both a and b part of the remaining tasks. To confirm the victim machine, I applied the filter **ip.dst == 192.168.0.1** helping me find out that around 8912 packets were sent to it, thus confirming it being the victim.

Now, coming to the last sub part of this question, to analyze the impact, I searched for other IP sources interacting with the server and I got 192.168.0.37. Then applying a filter to check the difference between the request sent and the response received, it was found out to be 2 s, while the same IP when interacting with another machine 115.0.0.2 receives response in 0.2 s, this actually shows the difference (Fig 8). Thus, the flood was successful in hampering the services of the server, slowing down the response time to nearly 10 times.

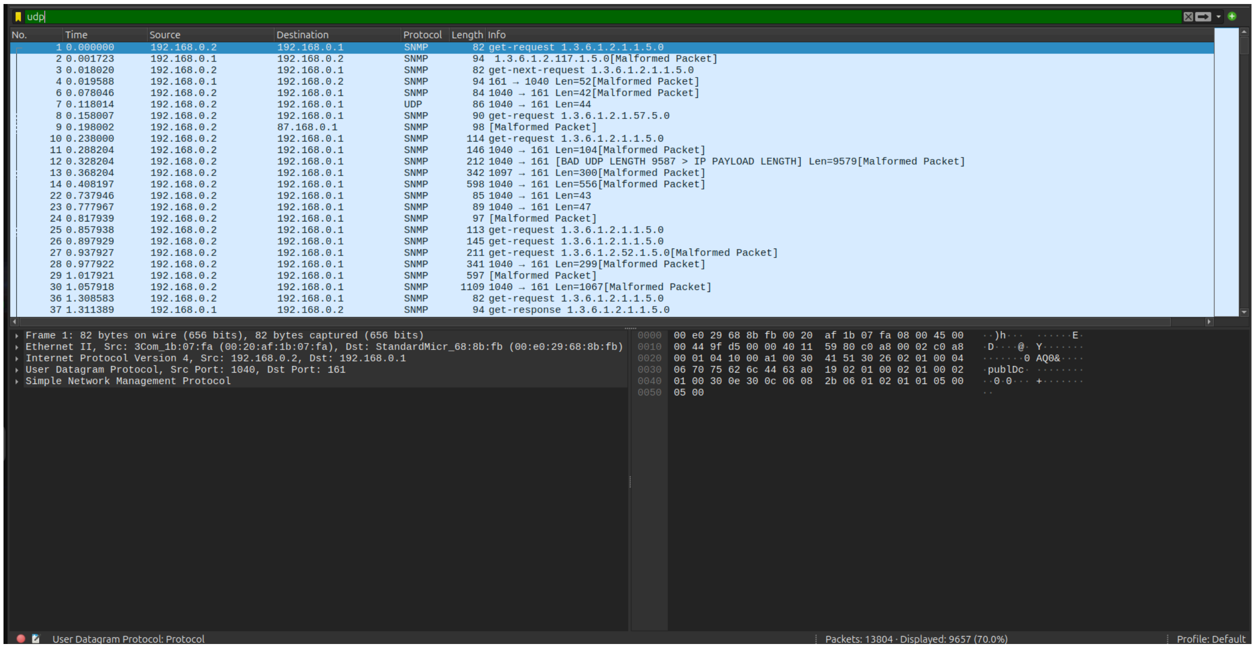

Fig 5: UDP Filter

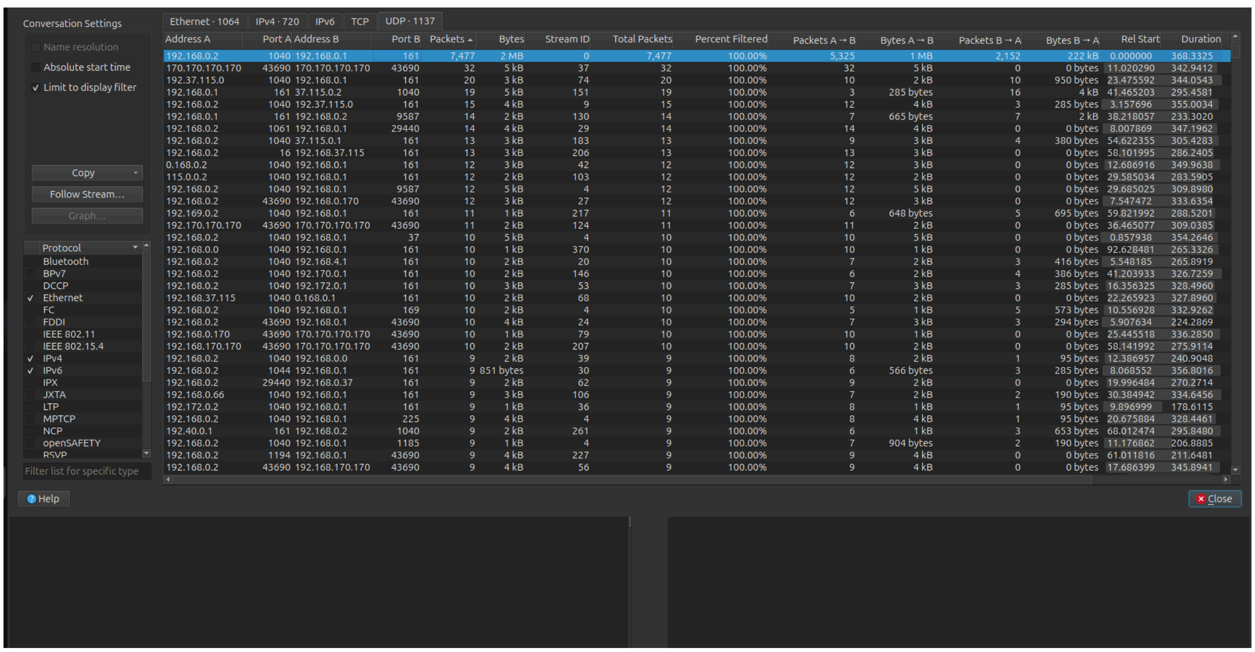

Fig 6: UDP attack statistics

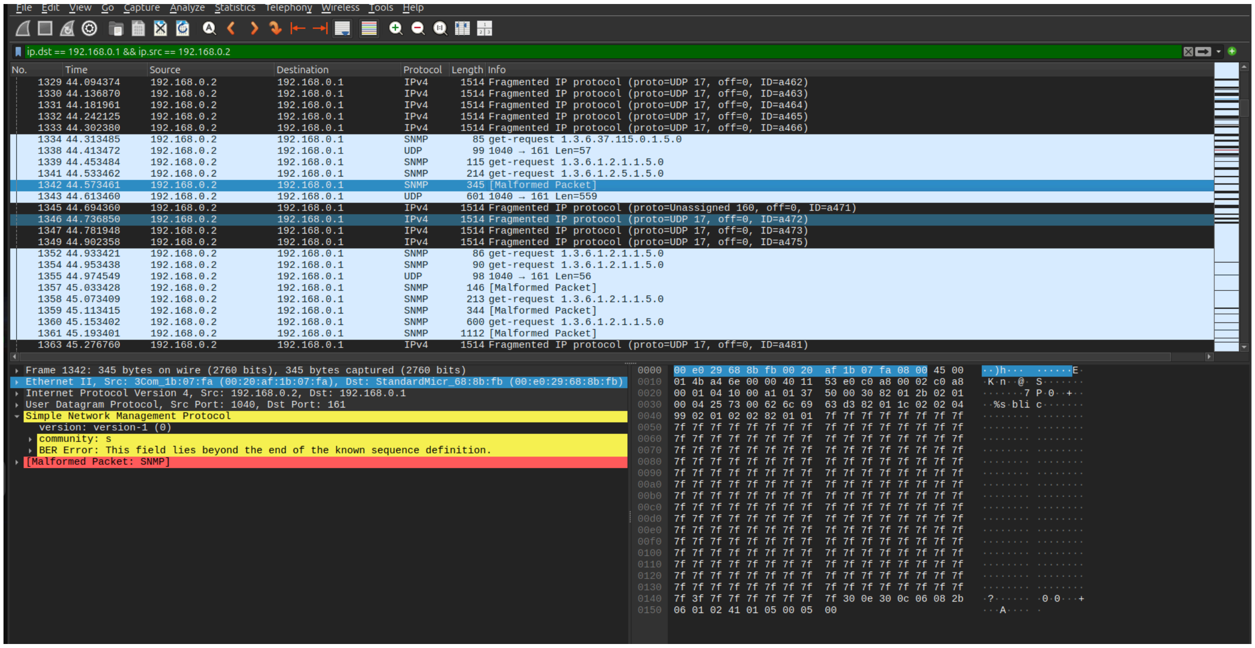

Fig 7: The Data of the packet is actually very much corrupted

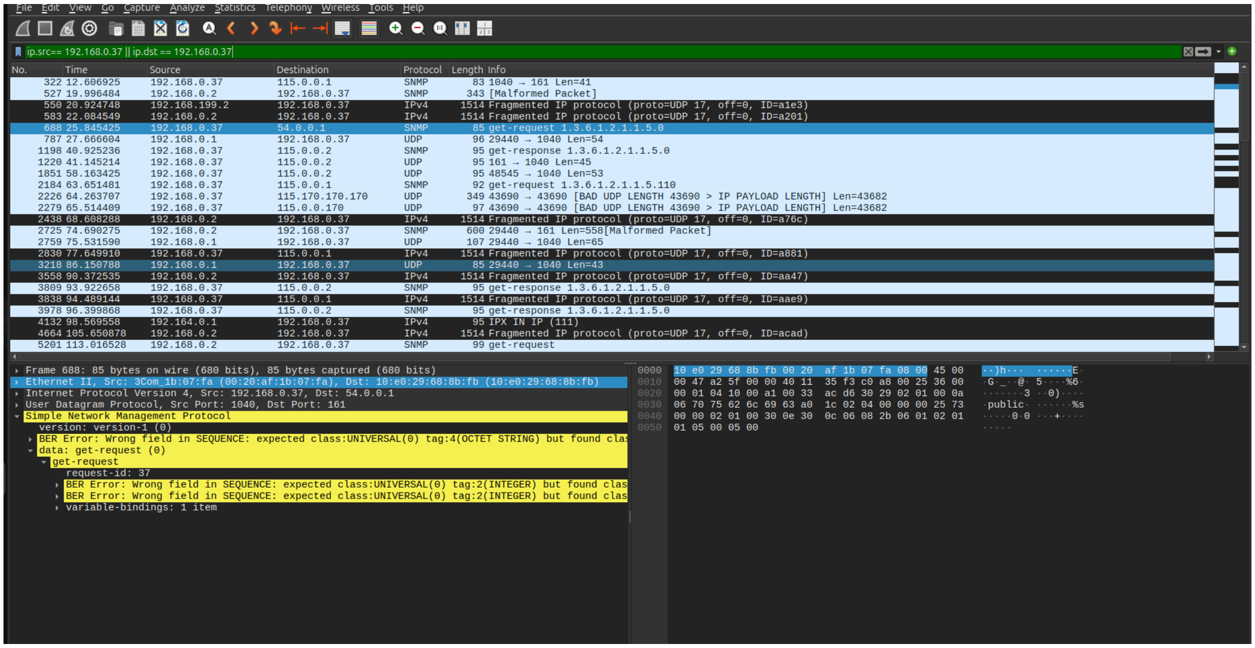

Fig 8: Impact of UDP flood