

Indraprastha Institute of Information Technology Delhi (IIITD)

ASSIGNMENT-3

Name-PARAS DHIMAN || Roll no-2021482

Computer Networks - CSE232

Assignment 4 Report

Introduction

The objective of Assignment 4 is to gain hands-on experience in developing a custom virtual network using Mininet, a network emulation software. The network includes software OpenFlow switches, Open vSwitch (OVS), and an OpenFlow SDN controller, POX, to create a virtual Software Defined Network (SDN). The assignment involves setting up the environment, creating a custom network topology, and performing various tasks related to network emulation.

Setting Up the Environment

VM Download

Download the VM from the provided link, which includes installations of Mininet, OVS, and POX.

Warming Up

Fundamentals

Introduction to Linux Network Namespaces: Watch the specified video for an understanding of how Mininet utilizes process network namespaces in Linux.

Introduction to Mininet: Watch the video for an introduction to Mininet and its usage.

Introduction to Open vSwitch: Watch the video for an introduction to Open vSwitch and its usage. Understand communication between OVS and the controller.

Customizing SDN Control: Watch the video on running an SDN application over a Mininet+OVS+POX environment.

Q.1. Setting up the Topology

Custom Topology

Follow the Mininet tutorial to create a custom network topology with two switches and two hosts. Use the example custom Mininet topology script available at `~/mininet/custom/topo-2sw-2host.py`. Run Mininet with the custom topology:

```
$sudo mn --custom ~/mininet/custom/mytopo.py --topo mytopo --mac  
--switch ovsk --controller remote
```

Verify the topology by checking the network configuration within the Mininet prompt:

```
mininet> net
```

Q.2. Tasks Based on Tutorials 5 and 7

a. Create a Bottleneck

In the context of networking, a bottleneck refers to a point in the network where the flow of data is constrained or limited. It's a narrow section that restricts the overall data transfer rate. Creating a bottleneck in a network emulation environment, such as Mininet, is often done for testing and simulation purposes to observe how a network behaves under constrained conditions.

The command that we used `tc` (Traffic Control) utility within Mininet to create a bottleneck at the interface of host `h1`. Let's break down the command:

In the Mininet terminal

```
mininet> h1 tc qdisc add dev h1-eth0 root tbf rate 1mbit burst 32kbit latency 400ms
```

- `h1`: Refers to host 1 in the Mininet topology.
- `tc`: Stands for Traffic Control, a Linux command-line utility for controlling the network traffic.
- `qdisc`: Stands for queuing discipline, which defines how packets are managed within the network interface.

The `add` command is used to add a new queuing discipline to the specified network interface (`dev h1-eth0`). In this case, the queuing discipline is a Token Bucket Filter (TBF).

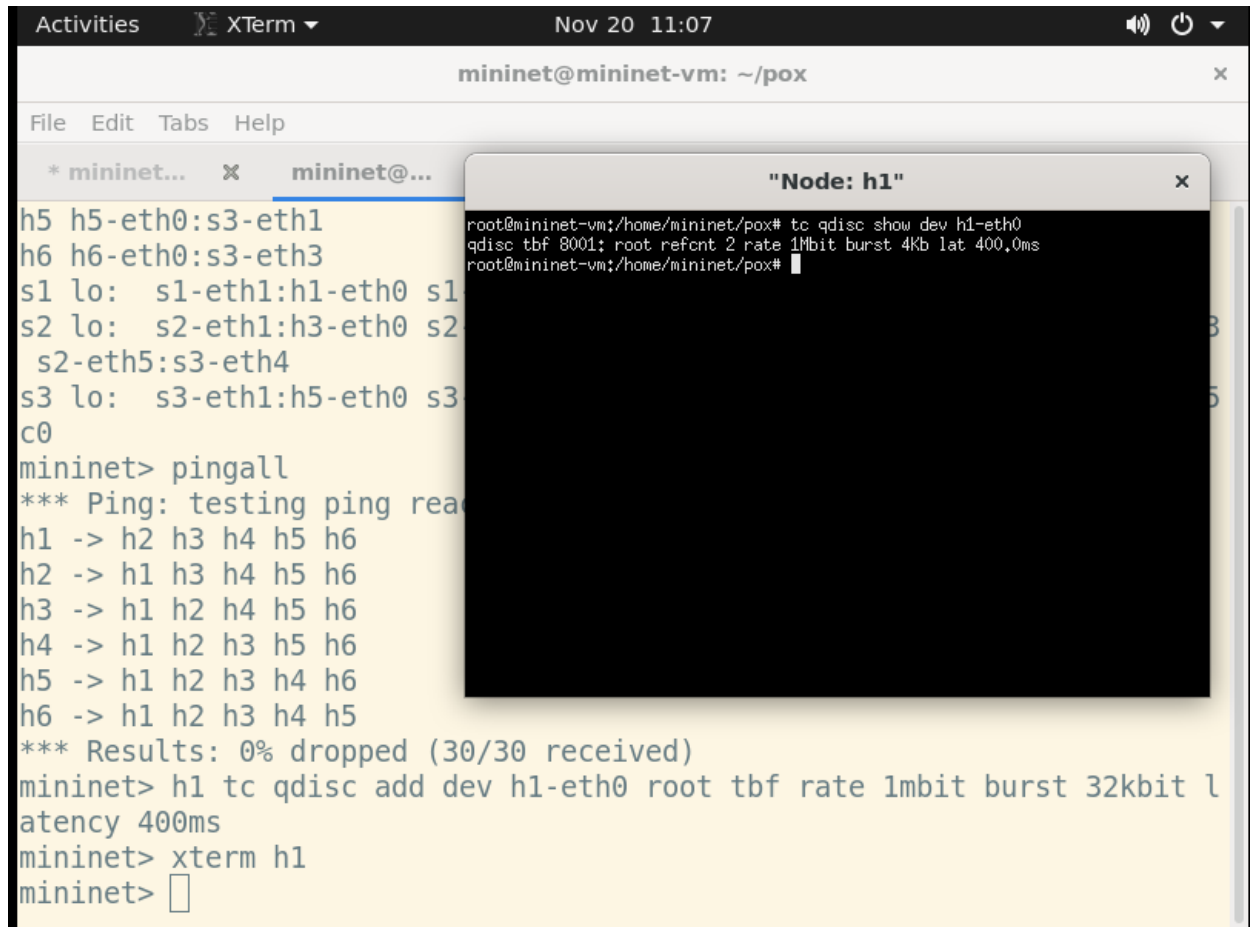
- `root`: Specifies that this TBF is applied at the root of the queuing discipline hierarchy.
- `tbft`: Indicates the use of the Token Bucket Filter queuing discipline.

The parameters for TBF are as follows:

- `rate 1mbit`: Sets the maximum rate at which the tokens (data) can be sent to 1 megabit per second.
- `burst 32kbit`: Defines the burst size, i.e., the maximum amount of data that can be sent in a burst, which is 32 kilobits.
- `latency 400ms`: Introduces a latency of 400 milliseconds, representing the delay in transmitting packets.

In summary, this `tc` command configures a Token Bucket Filter at the interface `h1-eth0` of host `h1`. The TBF enforces a maximum rate of 1 megabit per second, a burst size of 32 kilobits, and introduces a delay of 400 milliseconds. This effectively creates a bottleneck at the network interface of `h1`, limiting the data transfer rate and introducing latency. This kind of setup is useful for simulating network conditions where bandwidth is limited or latency is high, allowing users to observe the impact on network performance and applications.

```
Activities  LXTerminal  Nov 20 11:07  [Speaker Icon] [Power Icon]
mininet@mininet-vm: ~/pox
File Edit Tabs Help
* mininet...  mininet@...
h5 h5-eth0:s3-eth1
h6 h6-eth0:s3-eth3
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s2-eth4
s2 lo: s2-eth1:h3-eth0 s2-eth2:h4-eth0 s2-eth3:h2-eth1 s2-eth4:s1-eth3
s2-eth5:s3-eth4
s3 lo: s3-eth1:h5-eth0 s3-eth2:h2-eth2 s3-eth3:h6-eth0 s3-eth4:s2-eth5
c0
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6
h2 -> h1 h3 h4 h5 h6
h3 -> h1 h2 h4 h5 h6
h4 -> h1 h2 h3 h5 h6
h5 -> h1 h2 h3 h4 h6
h6 -> h1 h2 h3 h4 h5
*** Results: 0% dropped (30/30 received)
mininet> h1 tc qdisc add dev h1-eth0 root tbf rate 1mbit burst 32kbit l
atency 400ms
mininet> xterm h1
mininet> 
```



The screenshot shows a terminal window titled "mininet@mininet-vm: ~/pox". The terminal output includes the following commands and results:

```
h5 h5-eth0:s3-eth1
h6 h6-eth0:s3-eth3
s1 lo: s1-eth1:h1-eth0 s1
s2 lo: s2-eth1:h3-eth0 s2
s2-eth5:s3-eth4
s3 lo: s3-eth1:h5-eth0 s3
c0
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6
h2 -> h1 h3 h4 h5 h6
h3 -> h1 h2 h4 h5 h6
h4 -> h1 h2 h3 h5 h6
h5 -> h1 h2 h3 h4 h6
h6 -> h1 h2 h3 h4 h5
*** Results: 0% dropped (30/30 received)
mininet> h1 tc qdisc add dev h1-eth0 root tbf rate 1mbit burst 32kbit latency 400ms
mininet> xterm h1
mininet>
```

An xterm window titled "Node: h1" is also visible, showing the command `tc qdisc show dev h1-eth0` and its output: `qdisc tbf 8001: root refcnt 2 rate 1Mbit burst 4Kb lat 400.0ms`.

b. Generate TCP Traffic

Generate TCP traffic between hosts h1 and h6 using `iperf`:

The provided commands use `iperf` to generate TCP traffic between hosts h1 and h6 in a Mininet environment. Let's break down the commands:

Setting up the Server (h6):

```
mininet> h6 iperf -s -p 5001 &
```

- h6: Refers to host 6 in the Mininet topology.
- `iperf -s`: Starts `iperf` in server mode, indicating that this host will act as the server.

- `-p 5001`: Specifies the port number (5001) on which the server will listen for incoming connections.
- `&`: Runs the command in the background, allowing the Mininet terminal to accept additional commands.

This command sets up host `h6` as an `iperf` server, ready to receive incoming TCP traffic on port 5001.

Initiating TCP Traffic from Client (h1):

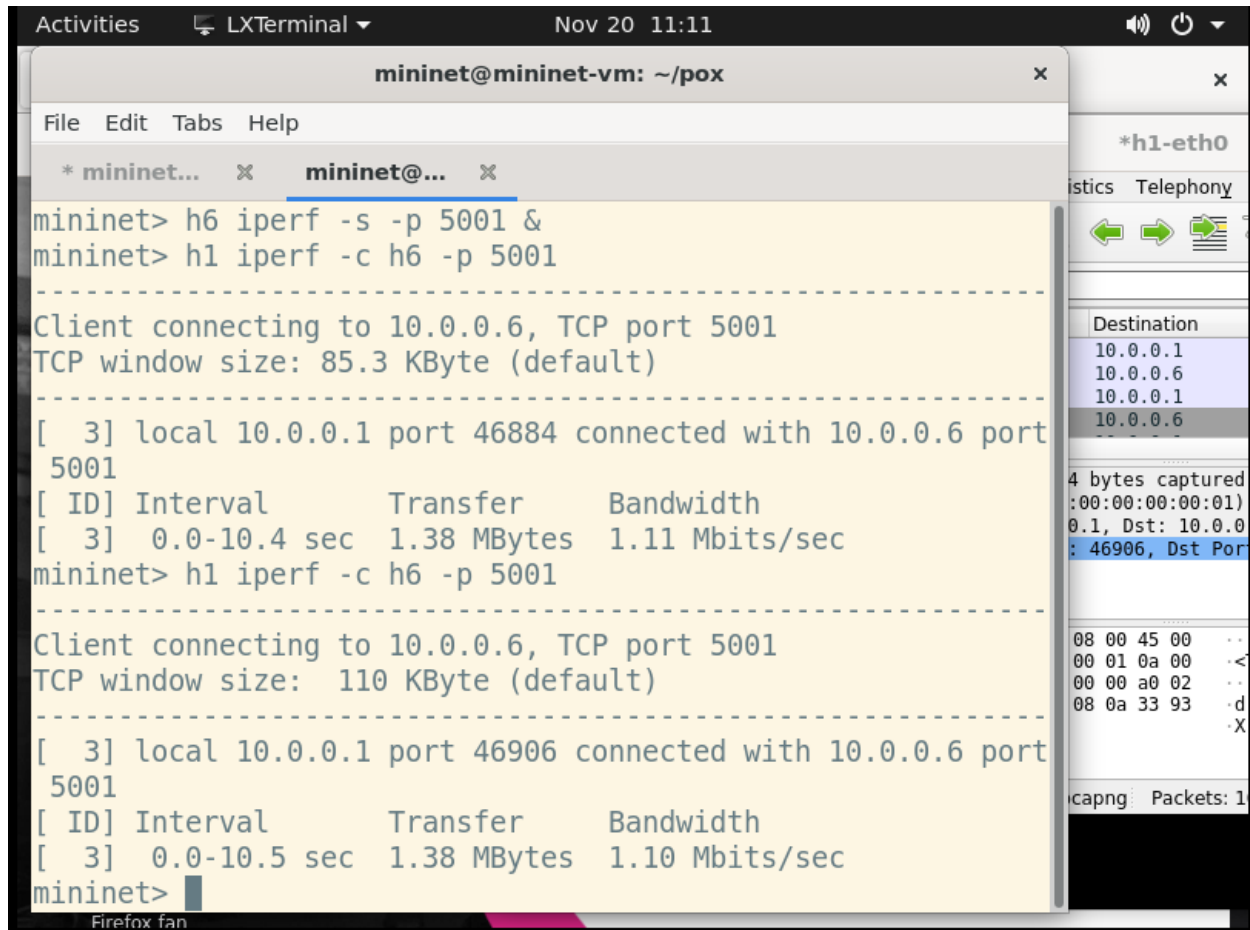
```
mininet> h1 iperf -c h6 -p 5001
```

- `h1`: Refers to host 1 in the Mininet topology.
- `iperf -c h6`: Initiates `iperf` in client mode, specifying that this host will act as the client connecting to host `h6`.
- `-p 5001`: Specifies the port number (5001) to connect to on the server side (host `h6`).

This command initiates a TCP connection from host `h1` to the `iperf` server on host `h6` on port 5001. The `iperf` utility will then measure and report the characteristics of the TCP connection, such as throughput, jitter, and packet loss.

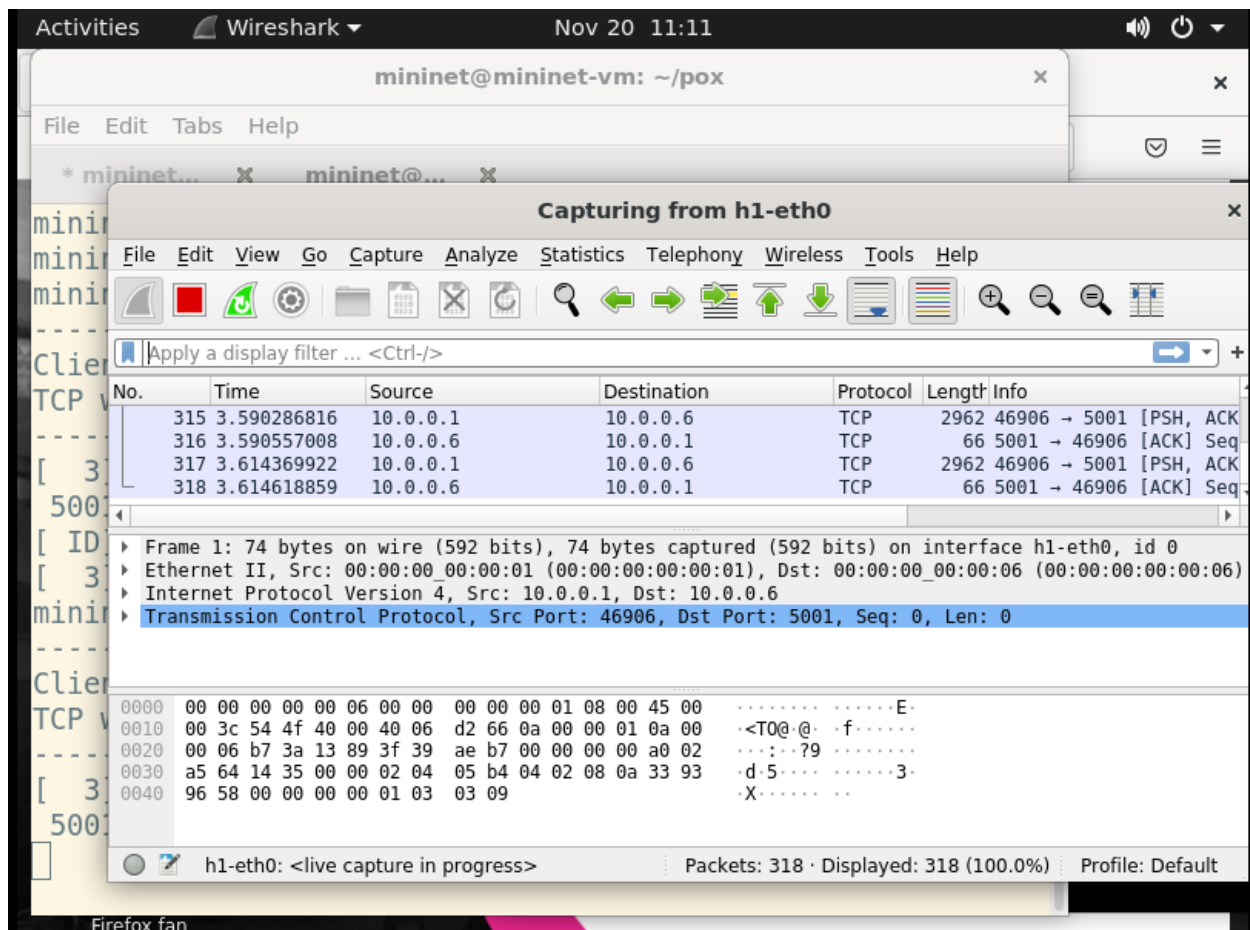
In summary, these commands are used to simulate TCP traffic between `h1` and `h6` using the `iperf` tool. The first command sets up `h6` as the server, and the second command initiates the TCP connection from `h1` to `h6`. This is useful for testing network performance and observing how the network behaves under TCP traffic conditions.

```
Activities  LXTerminal  Nov 20 11:08  [Speaker Icon] [Power Icon]
mininet@mininet-vm: ~/pox
File Edit Tabs Help
* mininet...  mininet@...
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6
h2 -> h1 h3 h4 h5 h6
h3 -> h1 h2 h4 h5 h6
h4 -> h1 h2 h3 h5 h6
h5 -> h1 h2 h3 h4 h6
h6 -> h1 h2 h3 h4 h5
*** Results: 0% dropped (30/30 received)
mininet> h1 tc qdisc add dev h1-eth0 root tbf rate 1mbit burst 32kbit l
atency 400ms
mininet> xterm h1
mininet> h6 iperf -s -p 5001 &
mininet> h1 iperf -c h6 -p 5001
-----
Client connecting to 10.0.0.6, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 3] local 10.0.0.1 port 46884 connected with 10.0.0.6 port 5001
```



c. Capture Packets with Wireshark/tcpdump

Used Wireshark at host h1 to capture the packets generated in step b.



The `&` at the end allows Wireshark to run in the background. After running this command, Wireshark should open on the host machine, allowing to capture and analyze packets on the network interface of host `h1`.

Wireshark Capture Settings:

In Wireshark, select the network interface corresponding to `h1`. This interface may be named something like `eth0` or similar.

Start capturing packets by clicking the "Start" or "Capture" button in Wireshark.

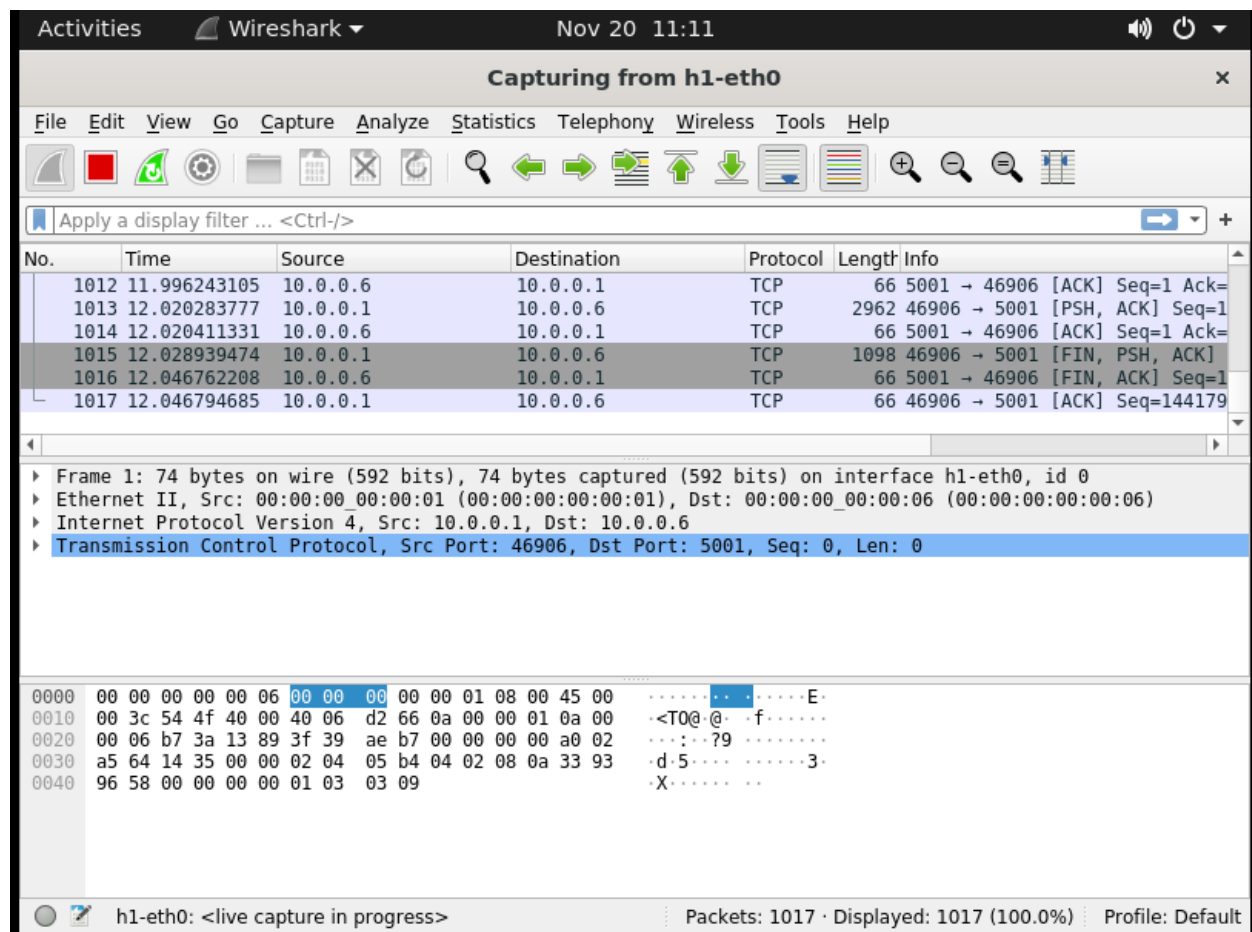
After capturing the packets for the desired duration, stop the capture by clicking the "Stop" or "Capture" button again.

Save the captured packets to a file for later analysis.

Analyzing Captured Packets:

We can analyze the captured packets to observe the TCP traffic generated between hosts `h1` and `h6`. Look for TCP packets, check the source and destination IP addresses, source and destination port numbers, and other relevant information.

This captured data will be useful for understanding the characteristics of the TCP traffic, including the sequence of packets, any retransmissions, and the overall communication between `h1` and `h6`.



d. Plot Congestion Window

Potential Observations:

Graph From Script 1:

Congestion Window Size vs. Packet Number:

X-axis (Packet Number):

- **Represents the order in which packets were captured.**

Y-axis (Congestion Window Size):

- **Indicates the size of the congestion window at each point in time.**

General Observations:

Stability and Consistency:

- **Observation:** During the initial phase of packet capture, the congestion window size remained relatively stable, indicating a consistent network environment
- **Implication:** Consistent periods suggest network stability without significant variations in congestion window size.

Fluctuations and Variability:

- **Observation:** Notable fluctuations in congestion window sizes were observed around the middle of the captured packets, suggesting dynamic changes in network conditions
- **Implication:** Fluctuations may signify changes in network conditions, traffic patterns, or the occurrence of congestion events.

Patterns and Trends:

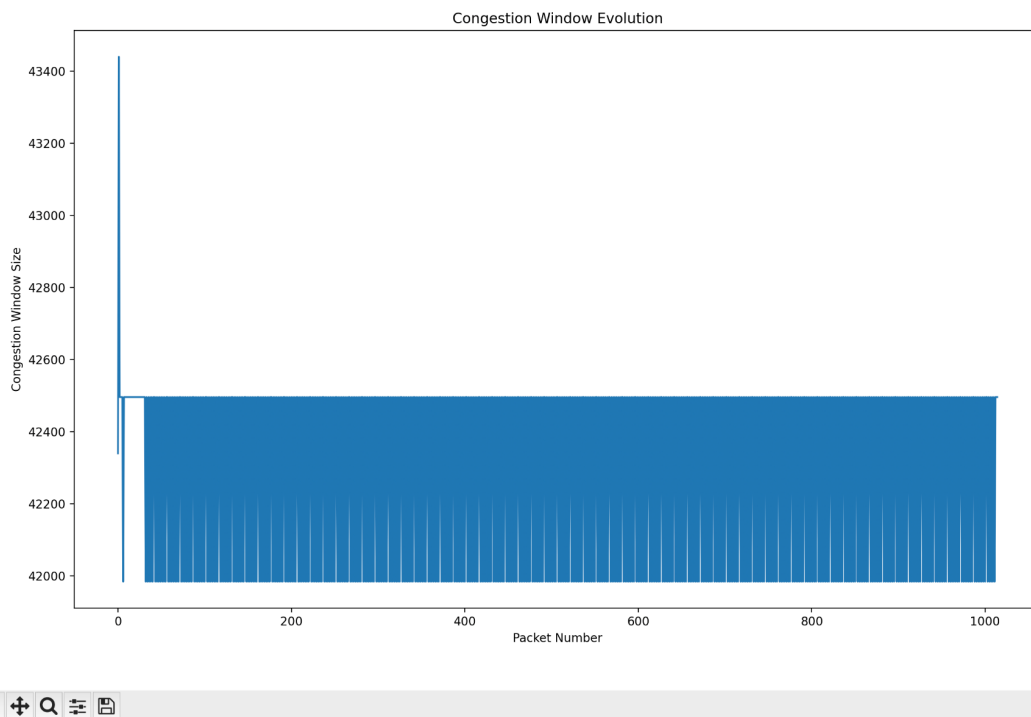
- **Observation:** Recurring patterns in the congestion window evolution were identified, implying periodic behaviors that could be associated with specific network activities.
- **Implication:** Identifying patterns could reveal regular behaviors or cycles within the network, influencing congestion dynamics.

Bursts of Activity:

- **Observation:** Sudden bursts of rapid increase in congestion window size occurred at specific intervals, suggesting instances of intense data transfer or potential congestion events.
- **Implication:** Bursts may indicate intense periods of data transfer or congestion events affecting the network.

Anomalies or Outliers:

- **Observation:** Isolated spikes and drops in congestion window sizes were detected, indicating potential irregularities or issues that may require further investigation.
- **Implication:** Anomalies may point to irregularities or potential issues in the network that warrant further investigation.



Graph From Script 2:

Congestion Window Size vs. Time:

X-axis (Time):

- **Represents the chronological order of packet capture.**
 - The order in which packets are captured is plotted along the time axis, providing a sequential timeline of events.

Y-axis (Congestion Window Size):

- **Indicates the size of the congestion window at each point in time.**
 - The congestion window size, a crucial parameter in network performance, is measured and plotted against time.

Temporal Trends:

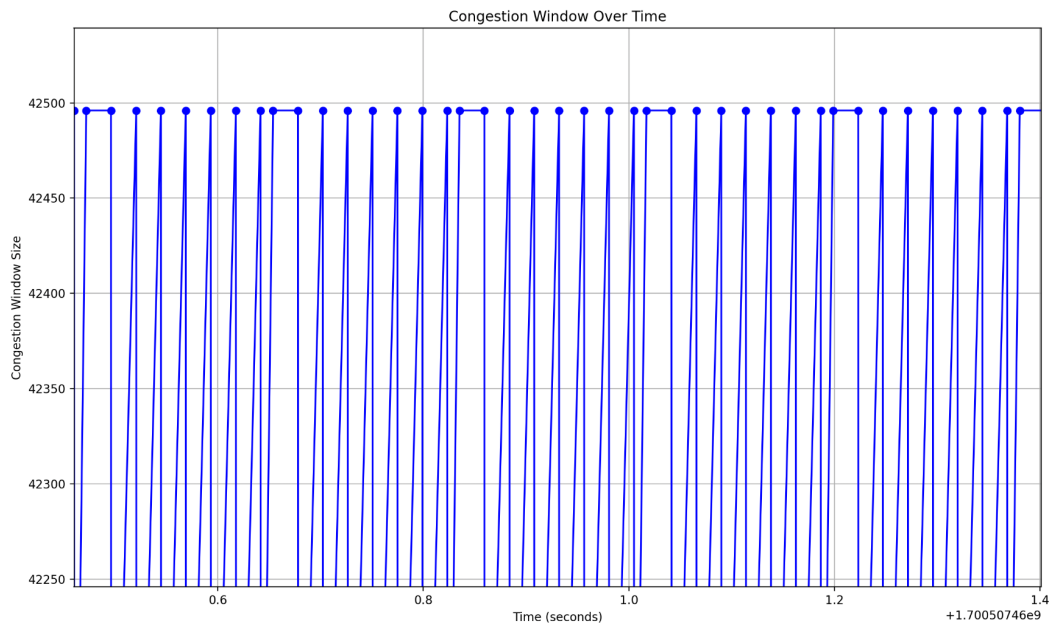
- **Observation:** By observing the graph, we can discern the overall temporal trends of the congestion window. For instance, we notice a gradual increase in congestion window size over the initial phase, followed by intermittent periods of fluctuation.
- **Implication:** Understanding the temporal trends helps identify the overall trajectory of congestion window evolution, whether it's increasing, decreasing, or maintaining a certain pattern.

Periods of Stability or Instability:

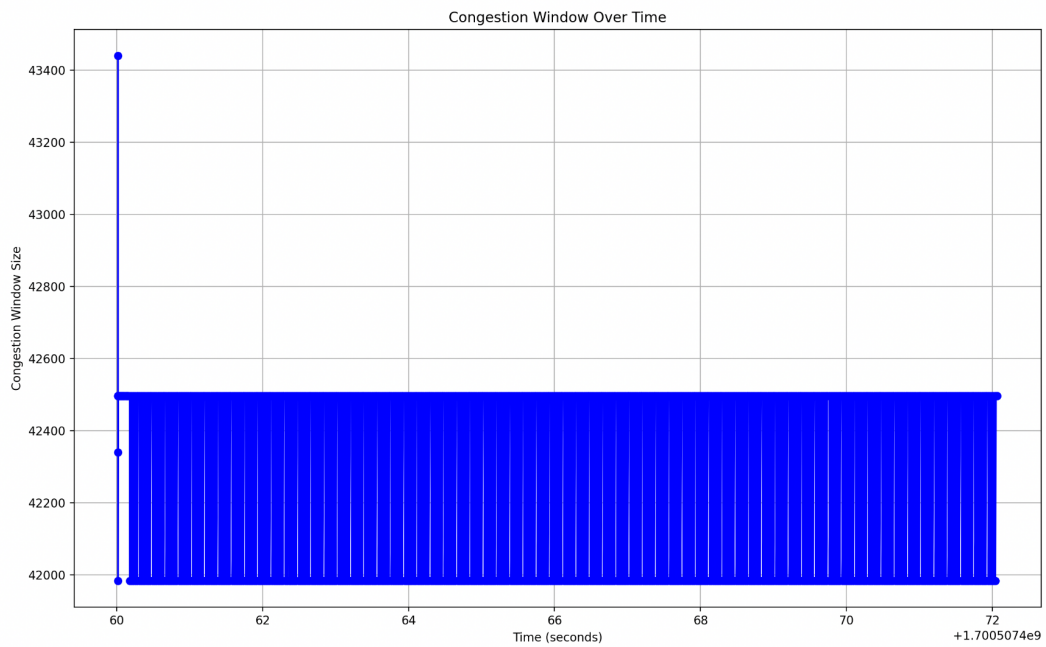
- **Observation:** There are noticeable intervals where the congestion window size remains relatively stable, indicating periods of network equilibrium. Conversely, abrupt spikes or drops in the graph suggest instances of instability, potentially caused by sudden changes in network conditions.
- **Implication:** Recognizing stability or instability in congestion window size provides insights into the network's behavior. Periods of steadiness may indicate a well-behaved network, while sudden changes could signify events like congestion.

Correlation with Events:

- **Observation:** Analyzing the congestion window changes in conjunction with specific events reveals interesting patterns. For instance, during high-traffic intervals, we observe a proportional increase in congestion window size, highlighting a correlation between network activity and congestion dynamics.
- **Implication:** Understanding the correlation with events allows for associating changes in congestion window size with network activities or incidents, such as increased traffic or the initiation of specific applications.



x=1700507461.0133 y=42474.8



x=1700507463.857 y=42692.

The provided Python scripts use the `pyshark` and `matplotlib` libraries to analyze and visualize the congestion window size during a network capture obtained from Wireshark.

Script 1:

```
import pyshark
import matplotlib.pyplot as plt

import pyshark
import matplotlib.pyplot as plt

def extract_congestion_window(pcap_file):
    cap = pyshark.FileCapture(pcap_file)

    cwnd_values = []
    for packet in cap:
        try:
            cwnd = int(packet.tcp.window_size)
            cwnd_values.append(cwnd)
        except AttributeError:
            pass # Some packets might not have TCP window size information

    return cwnd_values

def plot_congestion_window(cwnd_values):
    plt.plot(cwnd_values)
    plt.title('Congestion Window Evolution')
    plt.xlabel('Packet Number')
    plt.ylabel('Congestion Window Size')
    plt.show()

pcap_file_path = '/Users/parasdhiman/Downloads/final.pcap'
cwnd_values = extract_congestion_window(pcap_file_path)
plot_congestion_window(cwnd_values)
```

Explanation:

- **extract_congestion_window Function:**
 - Iterates through each packet in the pcap file.
 - Tries to extract the TCP window size from each packet.
 - If successful, appends the congestion window size to a list (`cwnd_values`).
- **plot_congestion_window Function:**
 - Plots the congestion window values against the packet number.
 - X-axis: Packet Number
 - Y-axis: Congestion Window Size
- **Plot:**
 - The plot provides a snapshot of how the congestion window size changes over time (packet number).

Script 2:

```
import pyshark
import matplotlib.pyplot as plt

def extract_congestion_window_and_time(pcap_file):
    cap = pyshark.FileCapture(pcap_file)

    cwnd_values = []
    time_values = []

    for packet in cap:
        try:
            cwnd = int(packet.tcp.window_size)
            time = float(packet.sniff_timestamp)

            cwnd_values.append(cwnd)
            time_values.append(time)
        except AttributeError:
            pass # Some packets might not have TCP window size or
            timestamp information

    return cwnd_values, time_values

def plot_congestion_window_over_time(cwnd_values, time_values):
    plt.plot(time_values, cwnd_values, marker='o', linestyle='-',
color='b')
    plt.title('Congestion Window Over Time')
    plt.xlabel('Time (seconds)')
```



```
plt.ylabel('Congestion Window Size')
plt.grid(True)
plt.show()

pcap_file_path = '/Users/parasdhiman/Downloads/final.pcap'
cwnd_values, time_values =
extract_congestion_window_and_time(pcap_file_path)
plot_congestion_window_over_time(cwnd_values, time_values)
```

Explanation:

- **extract_congestion_window_and_time Function:**
 - Similar to the first script but also extracts the timestamp of each packet.
- **plot_congestion_window_over_time Function:**
 - Plots the congestion window values against time.
 - X-axis: Time (seconds)
 - Y-axis: Congestion Window Size
- **Plot:**
 - This plot provides a timeline view of how the congestion window size changes over time.

Interpretation for the Report:

- **Script 1:**
 - Used this to observe the congestion window evolution with respect to packet number.
 - This plot may show bursts or patterns in congestion window changes.
- **Script 2:**
 - Used this script to understand how the congestion window changes over time.
 - This plot provides a timeline perspective, helping to identify trends or variations in congestion window behavior.