MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

MINI PROJECT REPORT OF COMPUTER NETWORKS LAB

PLOT OF MAXIMUM SEGMENT SIZE(MSS) FOR TCP-ORIENTED APPLICATION LAYER PROTOCOLS

SUBMITTED BY-

1) ISHITA PANDEY (REG.NO: 200905007)
2) SWAMIRAJU SATYA PRAVEEN VARMA (REG.NO: 200905044)
3) PALADUGU VENKATA YASHWANTH (REG.NO: 200905058)

ACKNOWLEDGEMENT

Our team (**Praveen Varma, Ishita Pandey and Yashwanth**) would like to thank the **Dept of CSE, MIT Manipal** for providing us with the necessary infrastructure and support to carry out our project.

We are grateful to our professor **Mr. Manoj R** who gave us an amazing chance to work on this project and for giving us valuable suggestions and ideas.

We express our gratitude to **Mr. Prashant Barla** for guiding us during our lab hours.

We would also like to thank our college for providing us with all the necessary resources for the project.

Finally, we would like to thank our acquaintances for always being with us and supporting us in every situation.

TABLE OF CONTENTS

ABSTRACT

TCP is a very commonly used and important network protocol as several abstract application layer protocols like **HTTP**, **SMTP** and **FTP** use **TCP** as their transport layer protocol. *The maximum segment size* (**MSS**) is a parameter of the options field of the TCP header that specifies the largest amount of data, in bytes, that a device can receive or send in a single TCP segment. This does not include the TCP header or the IP header length, and is a measure for only the data being sent. The Maximum Segment Size is set at the beginning of the connection within the TCP option area, by the TCP SYN packet during the TCP handshake. In this mini project we used **dpkt** module of python to parse the **pcap** file received from Wireshark, containing the packets of any application layer protocol and extract the MSS information from the SYN packets in the **pcap** file. We then plot a graph of the MSS vs relative time using **matplotlib** module of python in order to physically model the changes in MSS value over time and analyze the same.

LIST OF TABLES AND FIGURES

LIST OF ABBREVIATIONS

- **TCP** - Transmission Control Protocol
- **IP** - Internet Protocol
- **MSS** - Maximum Segment Size
- **HTTP** - Hyper Text Transfer Protocol
- **FTP** - File Transfer Protocol
- **SMTP** - Simple Mail Transfer Protocol

1 INTRODUCTION

1.1 GENERAL INTRODUCTION

The Transmission Control Protocol (**TCP**) provides a communication service at an intermediate level between an application program and the Internet Protocol, by providing a host-to-host connectivity at the transport layer of the Internet model. It is a **connection-oriented** protocol and requires that a connection be established between client and server before data is transmitted. TCP provides **reliable**, **ordered**, and **error-checked** delivery of a stream of data between sender and receiver. There are three phases that the protocol's operations may be divided into:

- The Connection establishment phase
- The data transfer phase
- The connection termination phase

The **maximum segment size (MSS)** is the largest amount of data, specified in bytes, that a host device can receive in a single TCP segment. To avoid IP **fragmentation**, which can lead to packet loss and excessive retransmissions, the MSS should be set **small enough**. To accomplish this, the MSS is announced by each side using the **MSS option** when the TCP connection is established in the SYN packet. Throughout the connection, the TCP implements its **end-to-end congestion-control mechanism**. It does not use any network-assisted congestion control because the IP layer does not provide any feedback to the end systems regarding congestion. It detects congestion by the occurrence of a **loss event** (signaled by a timeout or triple duplicate acknowledgment) and then implements the congestion control algorithm, consisting of slow start, congestion avoidance and fast recovery. The sender side TCP maintains a track of the **congestion window(cwnd)** to limit the amount of unacknowledged data; therefore, the rate of transmission is roughly cwnd/RTT bytes/second. The cwnd is manipulated at each of the three states, decreased in the case of a loss event and increased when a new acknowledgment is received, for which it is said to be a **self-clocking mechanism**.

In the slow start state it increases the cwnd by 1 MSS for every acknowledgement, causing the transmission rate to grow exponentially. In the case of a loss event, the cwnd is lowered to 1 MSS and the slow start threshold value(ssthresh) is set to half the value of cwnd when the loss event occurred. When the cwnd increases and reaches the ssthresh value, congestion avoidance begins where the cwnd is increased linearly at a rate of 1 MSS per RTT. If the loss event occurrs due to a triple duplicate acknowledgment the response is a little less drastic and TCP enters the fast recovery phase where cwnd is increased by 1 MSS for every duplicate acknowledgement. These changes in cwnd can be monitored by observing the changes in MSS throughout the connection in order to understand how the congestion control mechanism changes its rate to accommodate the traffic in the network.

1.2 REQUIREMENTS

i) HARDWARE

- At least 500 MB of available disk space.
- 64-bit AMD64/x86-64 or 32-bit x86 CPU architecture.
- At least 2GB RAM

ii) SOFTWARE

- Any version of Python after Python 2.0 installed
- Wireshark installed
- Windows 7 above or Ubuntu 16.04+
- **Modules and tools:**
  - **dpkt python module:** A python module for fast, simple packet creation / parsing, with definitions for the basic TCP/IP protocols.
  - **Matplotlib:** A plotting library for the Python programming language.
  - **Wireshark:** A network protocol analyzer that captures packets from a network connection.

2 PROBLEM DEFINITION:

Devise a software program to model the MSS values for TCP-Oriented Application Layer Protocols and plot the change in MSS with time, along with an analysis of the same.

3 OBJECTIVES:

- To capture packets from networks running on an end system through Wireshark.
- To extract MSS values from the **pcap** file, which contains all information about the TCP packets captured using Wireshark, by implementing the **dpkt** module of Python.
- To plot the graph of **MSS vs Time** using **matplotlib** library of Python.
- To analyze the output obtained based on theoretical predictions. Allow the user to gauge whether congestion in their network is extremely high, low or optimum.

4 METHODOLOGY:

1. Open any web browsers from which network packets can be captured.
2. Open Wireshark and set it to capture ethernet datagrams. Allow it to run until a substantial number of packets have been captured. This will be seen in the output window of Wireshark. Now the TCP packets of application layer protocols like HTTP have been captured.
3. Stop the capture and download the pcap file of all packet information from Wireshark.
4. Run the python code as shown below by entering the name of the pcap that has been saved. The program will extract MSS values from **SYN** packets using the **dkpt** module of Python. Only the SYN packets in **TCP** connections contain the MSS in the **options** field, which

will be recorded.

5. The program will also extract **timestamp** from the packets along with the MSS and convert it **HH:MM:SS** and consequently to seconds.
6. Final graph of the MSS with respect to time, will be plotted using the **matplotlib** library. The exact data points will be shown in terminal based on which the user can conduct their analysis to see whether the congestion is very high, low or optimum.

5 IMPLEMENTATION DETAILS

We have captured traffic from the network connection of our laptop using Wireshark and filtered the captured packets for only application layer protocols that use TCP. This file was then downloaded as a pcap file and passed as the input to the python script below in order to plot the graph of MSS vs time. The **dpkt** module has a **pcap.Reader()** class that is a pcap-compatible pcap file reader using which we start parsing the pcap files. The **pcap.Reader()** class returns a series of tuples containing the corresponding timestamp and data. We then iterate over the tuples to extract useful information. First the data from the ethernet packet of the data link layer is extracted, providing the IP packet of the network layer. In this IP packet, if the protocol field contains the number 6, it indicates that the packet inside is a TCP payload, allowing such packets to get processed further.

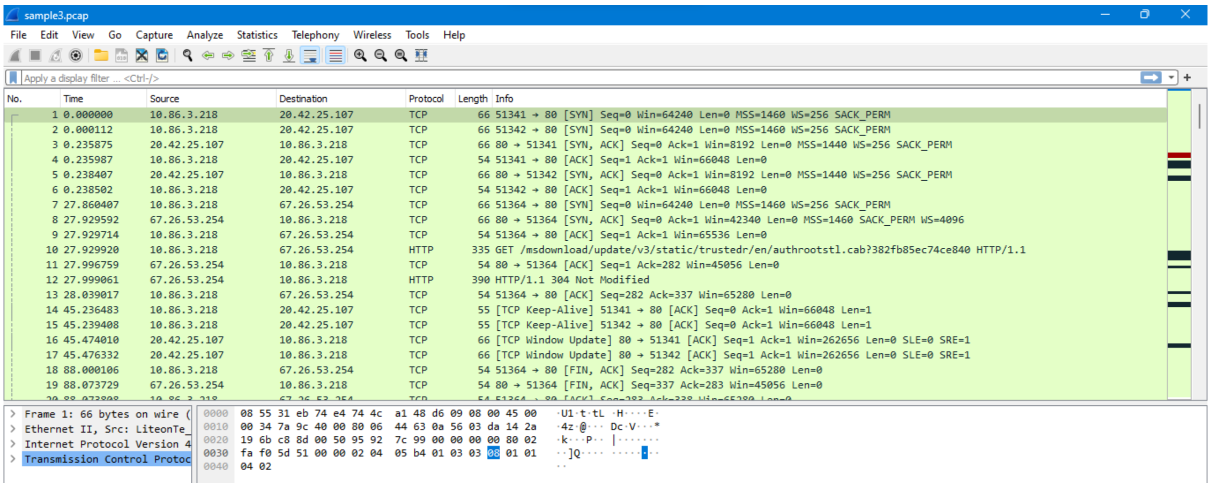
Next data is extracted from the IP packet, bringing for the TCP packet of the Transport layer. From the options field of the TCP packet, if MSS size is present then it is obtained using the **dpkt.tcp.parse_opts** method. The **dpkt.tcp.parse_opts** method returns a list containing [option number, data]. A value of 2 in the options field indicates the presence of MSS in the option field in the TCP header. So, if **option [0]** is 2 then the corresponding data field **option[1]** is unpacked, which returns a tuple and the 0th index of the tuple contains the MSS value which is then stored in a list. From the same packet that was used to get MSS, the timestamp is then used to get the time in seconds. First the timestamp is converted to UTC format, then to proper HH:MM: SS format, which is then converted to seconds and appended to a list for plotting of the graph. After both the **mss** list and **time** list are obtained, a graph is plotted using the **matplotlib.pyplot.plot()** function and **matplotlib.pyplot.scatter()** shown in the figures below.

Code:

```
import dpkt
import os
import sys
import socket
import datetime
import struct
from  datetime import datetime as dtime
import matplotlib.pyplot as plt

#method to get the MSS size from the options field of TCP header
def get_MSS(options) :
    options_list = dpkt.tcp.parse_opts ( options )      #Parse TCP option
buffer into a list of (option, data) tuples
    for option in options_list :
        if option[0] == 2 :
            mss = struct.unpack(">H", option[1])        #extracting the
mss value
            return mss[0]

#Method to extract the payload prsent in the  packets of different
layers
def parsePcap(pcap):
    mss=[]      #initializing mss and time arrays to extract and store
values of each tcp segment
    time=[]
    cnt = 0
    pkt_cnt = 0
    for (ts,buf) in pcap:       #ts as timestamp and buffer, traversing
over each packet in pcap file

        try:
            eth = dpkt.ethernet.Ethernet(buf)       #unpacking the
ethernet frame
            ip = eth.data                           #getting the data
within the ethernet frame(ip packet)
            if ip.p==dpkt.ip.IP_PROTO_TCP:      #checking if it is a tcp
packet
                tcp = ip.data                   #set the tcp data
                pkt_cnt += 1
                m = get_MSS(tcp.opts)           #send the options part of
the tcp packet to extract mss from
                if m==None:
                    cnt+=1
                    continue
                    # mss.append(m)
                mss.append(m)           #appending newest  mss value to
the array
                temp = str(datetime.datetime.utcfromtimestamp(ts))
                time_string = temp[11:19]
                date_time = dtime.strptime(time_string, "%H:%M:%S")
                a_timedelta = date_time - datetime.datetime(1900, 1, 1)
                seconds = a_timedelta.total_seconds()
                time.append(seconds)                    #appeding
according time into the time array in seconds
        except Exception as e:
            print("Error!",e)
            sys.exit(1)

    #IF no packet found with MSS or no SYN packet found
    if cnt==pkt_cnt:
        print("No packets found with MSS info")
        sys.exit(1)
    print("Total number of mss values recorded are ")
    print(len(time))
    print("Total number of time values recorded are " )
    print(len(mss))
    t = time[0]
    for i in range(len(time)):
        time[i] -= t

    print("MSS:",*mss)
    print("Time:",*time)
    #plot graph
    plt.plot(time,mss,color='b')
    plt.scatter(time,mss,color='k')
    plt.title('MSS vs Time (seconds)')
    plt.xlabel('Time (seconds)')
    plt.ylabel('MSS')
    plt.show()

def main():
    file_name = input("Enter the pcap filename or pathname: ")
    try:
        f = open(file_name,'rb')
        pcap = dpkt.pcap.Reader(f)      #sending the file to be read by
the pcap reader
    except Exception as e:
        print("Error!",e)
        sys.exit(1)
    try:
```

```python
        parsePcap(pcap)     #sending pcap variable into parsing function
    except Exception as e:
        print("Error!",e)

if __name__== "__main__":
    main()
```

OUTPUT:


sample3.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time       Source         Destination    Protocol Length Info
  1 0.000000   10.86.3.218    20.42.25.107   TCP       66 51341 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
  2 0.000112   10.86.3.218    20.42.25.107   TCP       66 51342 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
  3 0.235875   20.42.25.107   10.86.3.218    TCP       66 80 → 51341 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1440 WS=256 SACK_PERM
  4 0.235987   10.86.3.218    20.42.25.107   TCP       54 51341 → 80 [ACK] Seq=1 Ack=1 Win=66048 Len=0
  5 0.238407   20.42.25.107   10.86.3.218    TCP       66 80 → 51342 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1440 WS=256 SACK_PERM
  6 0.238502   10.86.3.218    20.42.25.107   TCP       54 51342 → 80 [ACK] Seq=1 Ack=1 Win=66048 Len=0
  7 27.860407  10.86.3.218    67.26.53.254   TCP       66 51364 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
  8 27.929592  67.26.53.254   10.86.3.218    TCP       66 80 → 51364 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1460 SACK_PERM WS=4096
  9 27.929714  10.86.3.218    67.26.53.254   TCP       54 51364 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
 10 27.929920  10.86.3.218    67.26.53.254   HTTP     335 GET /msdownload/update/v3/static/trustedr/en/authrootstl.cab?382fb85ec74ce840 HTTP/1.1
 11 27.996759  67.26.53.254   10.86.3.218    TCP       54 80 → 51364 [ACK] Seq=1 Ack=282 Win=45056 Len=0
 12 27.999061  67.26.53.254   10.86.3.218    HTTP     390 HTTP/1.1 304 Not Modified
 13 28.039017  10.86.3.218    67.26.53.254   TCP       54 51364 → 80 [ACK] Seq=282 Ack=337 Win=65280 Len=0
 14 45.236483  10.86.3.218    20.42.25.107   TCP       55 [TCP Keep-Alive] 51341 → 80 [ACK] Seq=0 Ack=1 Win=66048 Len=1
 15 45.239408  10.86.3.218    20.42.25.107   TCP       55 [TCP Keep-Alive] 51342 → 80 [ACK] Seq=0 Ack=1 Win=66048 Len=1
 16 45.474010  20.42.25.107   10.86.3.218    TCP       66 [TCP Window Update] 80 → 51341 [ACK] Seq=1 Ack=1 Win=262656 Len=0 SLE=0 SRE=1
 17 45.476332  20.42.25.107   10.86.3.218    TCP       66 [TCP Window Update] 80 → 51342 [ACK] Seq=1 Ack=1 Win=262656 Len=0 SLE=0 SRE=1
 18 88.000106  10.86.3.218    67.26.53.254   TCP       54 51364 → 80 [FIN, ACK] Seq=282 Ack=337 Win=65280 Len=0
 19 88.073729  67.26.53.254   10.86.3.218    TCP       54 80 → 51364 [FIN, ACK] Seq=337 Ack=283 Win=45056 Len=0

> Frame 1: 66 bytes on wire (
> Ethernet II, Src: LiteonTe_
> Internet Protocol Version 4
> Transmission Control Protoc

0000  08 55 31 eb 74 e4 74 4c  a1 48 d6 09 08 00 45 00   ·U1·t·tL ·H····E·
0010  00 34 7a 9c 40 00 80 06  44 63 0a 56 03 da 14 2a   ·4z·@··· Dc·V···*
0020  19 6b c8 8d 00 50 95 92  7c 99 00 00 00 00 80 02   ·k···P·· |·······
0030  fa f0 5d 51 00 00 02 04  05 b4 01 03 03 08 01 01   ··]Q···· ········
0040  04 02                                              ··
```

Figure 1: Sample data collected from Wireshark capture of HTTP packets

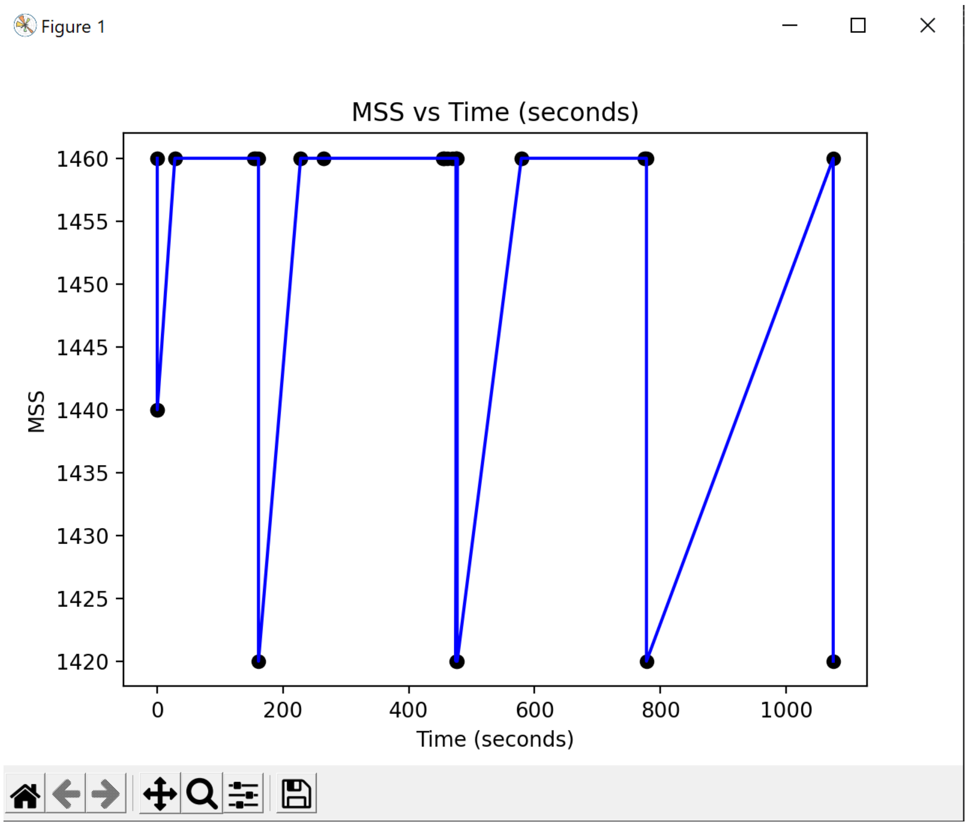
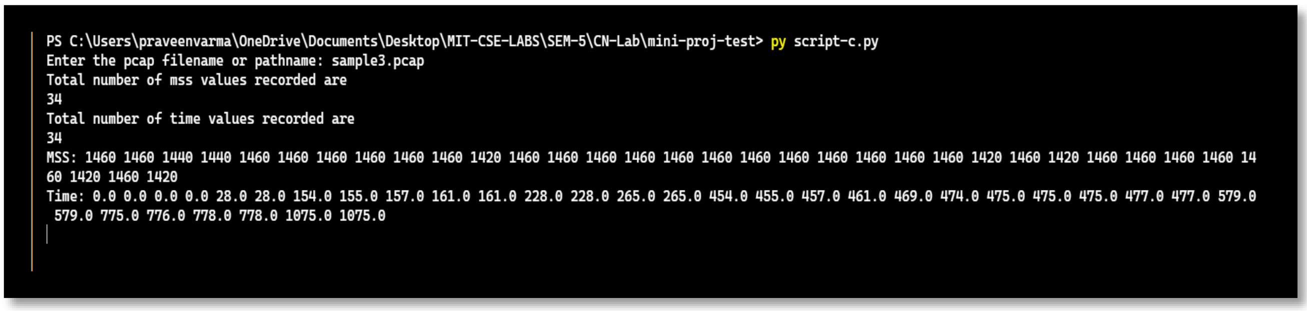


Figure 2: Graph of MSS vs Time(s) bases on Wireshark output



```
PS C:\Users\praveenvarma\OneDrive\Documents\Desktop\MIT-CSE-LABS\SEM-5\CN-Lab\mini-proj-test> py script-c.py
Enter the pcap filename or pathname: sample3.pcap
Total number of mss values recorded are
34
Total number of time values recorded are
34
MSS: 1460 1460 1440 1440 1460 1460 1460 1460 1460 1460 1420 1460 1460 1460 1460 1460 1460 1460 1460 1460 1460 1460 1460 1420 1460 1420 1460 1460 1460 1460 14
60 1420 1460 1420
Time: 0.0 0.0 0.0 0.0 28.0 28.0 154.0 155.0 157.0 161.0 161.0 228.0 228.0 265.0 265.0 454.0 455.0 457.0 461.0 469.0 474.0 475.0 475.0 475.0 477.0 477.0 579.0
 579.0 775.0 776.0 778.0 778.0 1075.0 1075.0
```

Figure 3: Terminal logs of MSS and Time recorded per packet

Analysis of the Output Obtained:

By analyzing the output graph in Figure 2, it is clear that the maximum MSS value never goes higher than 1460 and the minimum MSS value seen is 1420. Throughout the time the packets are captured, the MSS value oscillates between these two numbers, and sometimes stays constant at 1460 for a significant period of time. It can be understood that the drops in MSS seen at time periods of approximately 180, 480, 790 and 1100 seconds occur due to congestion in the network. This scenario occurs in case of traffic overloading when the link handles data in a much greater amount than its capacity.

There are a few major reasons which may have caused the congestion to occur. Firstly, the **Border Gateway Protocol** does not consider the amount of traffic already present while routing a packet, making it possible that all packets are being routed on the same link, causing congestion. In addition, a frequent use of the device at its maximum reported capacity causes over utilization and it becomes unable to maintain its theoretically assigned capacity, causing packet loss and data bottlenecks. **Over-subscription** and the use of the server by several other external clients can also cause congestion because it may be consistently slower or faster at different times of the day based on its usage, especially in case of **broadcast storms** where the requests for a network increase suddenly. **Unsolicited traffic** like advertisements or cached junk data can also lead to network congestion, hindering the useful segments from reaching the user. It is also necessary for a network to be configured to meet the needs of the user and be physically designed to handle the right load. In case of deficiency, for example a large network with **poor subnet management** that has not been scaled according to requirements, can cause serious congestion in the network because it is physically incapable of handling the required load. Other than these, **antiquated hardware** and **devices requiring updates** can also lead to congestion and low throughput in the network. Any of these may have caused the congestion seen in the graph above but the most likely causes are over-subscription and unsolicited traffic.

TCP detects these scenarios of congestion through its practice of **bandwidth probing** in which it increases its transmission rate in response to successful acknowledgements and lowers the rate when a loss event occurs. This cycle can be seen clearly in Figure 2 as the sender attempts to transmit packets at a high rate for as long as it can, decreases the MSS when a loss event occurs, and then again increases the rate of transmission once it has backed off.

When observed mid-connection, TCP performs congestion control by performing linear increase in its congestion window of 1 MSS per RTT and decreases the cwnd by half in case of a triple duplicate acknowledgement which signals congestion, controlling the transmission rate accordingly. This theoretical form of **additive increase, multiplicative decrease** is proved by the practical output above. Overtime it creates a pattern as seen in the graph in Figure 4. This algorithm is the basis of why TCP is considered a fair protocol as the bandwidth provided to each host is approximately R/K, where R is transmission rate in bps and K is the number of TCP connections.

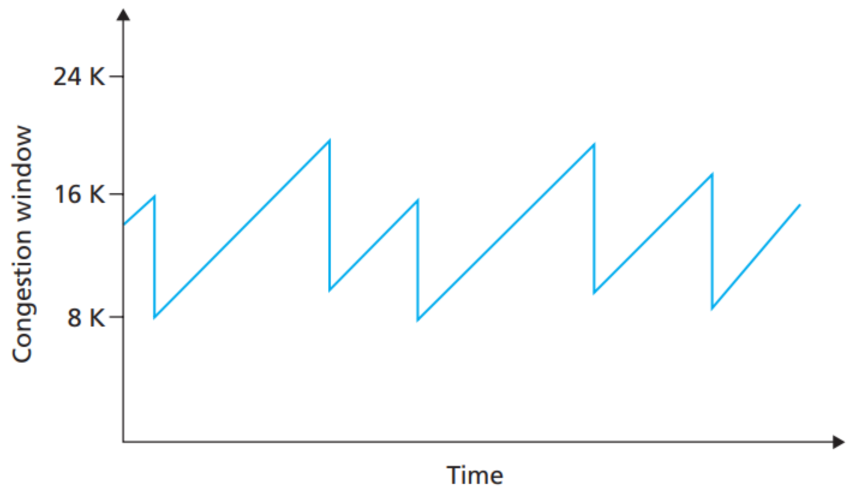


Figure 4. Theoretically expected graph of trend in Congestion Window over Time

5 CONTRIBUTION SUMMARY

Praveen: Researched about various packet parsing libraries in Python and implemented **dpkt** library, which is a python module for fast, simple packet creation/parsing, with definition for the basic TCP/IP protocols. Also contributed to method to get the MSS size from the options field of TCP header.

Ishita: Deep dive into Wireshark, a free and open-source packet analyzer and learnt about various filters in packet capture and generated multiple **pcap** files to test multiple cases. Also helped with implementing the method to extract the payload present in the packets of different layers.

Yashwanth: Have done research about various visualization libraries in Python and implemented the **matplotlib** which is a comprehensive library for creating static, animated, and interactive visualizations in Python. Integrated all the methods written in the main method and performed debugging by running multiple cases.

All our results are combined whilst preparing the **Report** for the project.

7 References:

- James F. Kurose & Keith W. Ross, Computer Networking A Top-Down Approach, (6e), Pearson Education,2013
- https://www.geeksforgeeks.org/what-is-network-congestion-common-causes-and-how-to-fix-them/
- https://en.wikipedia.org/wiki/Maximum_segment_size
- https://en.wikipedia.org/wiki/Transmission_Control_Protocol
- https://dpkt.readthedocs.io/en/latest/api/index.html
- https://datatracker.ietf.org/doc/html/rfc793
- https://datatracker.ietf.org/doc/html/rfc791#page-11
- https://readthedocs.org/projects/dpkt/downloads/pdf/latest