

Task 1: Application Layer and NS-3 Basics

Goal:

Part 1 and 2: Study and understand application layer protocols using Wireshark and other tools.

Part 3: Run your first network simulation program in NS-3 and understand the basics.

Part 1

1. When you browse IIT Bhilai main page how many get request is sent (how many of the get request are for embedded content and how many get requests for the text)? Plot the IO graph for packets sent to iitbhilai.ac.in and packets received from iitbhilai.ac.in
2. For the response to your HTTP get request, get the image reconstructed by hex editor.
3. Find the interpacket interval between multiple get requests.
4. Find the throughput observed while browsing the IIT Bhilai site under two cases
 - a. When no other traffic in the background
 - b. When a large file download is going.

the throughput calculation needs filtering only IIT Bhilai pages (from the get request originated from your browser till the last response has arrived at end of the web page)

Part 2

1. Surf a website (other than google.com) of your choice and discuss the end-to-end process of web page loading using Wireshark. How much time does it take to load the page? Find out how many connections are used to download this page, are these connections persistent or non-persistent? How many objects have been transferred on these connections? Which object took the longest time to download?
2. Find all the active TCP ports on your system. Identify the ports and PIDs of your web browser. Can you identify the port number and PID of specific TAB in your browser? Find out if any of the services running in your system uses the standard ports of HTTP, DHCP, DNS, SMTP, and FTP.
3. The root server on the Internet is in domain root-servers.net. You can see the list of all root servers using ***dig*** [DNS lookup utility] or using any tool/command.

Use dig to ask the root server the address of www.iitbhilai.ac.in, without recursion. Go through the hierarchy from the root without recursion, following the referrals manually, until you have found the address of www.iitbhilai.ac.in

List all the name servers involved to find out the IP address of the www.iitbhilai.ac.in?

Do the same exercise for 2 more websites with different top-level domains (.com, .edu, .org, etc.)

Part 3

1. Modify the Demo.cc code so that a total of X messages is sent by the client, one every 1 second, where X is the number of letters in your last name. Compile and run the script and see if the result is what you expected. Compare the throughput observed before and after modification of Demo.cc code and justify the throughput observed in both cases.
2. In Demo.cc, Instead of echo client-server app, transmit 1MB file from node 0 to node 1 and record the observed throughput. In which case (echo client-server vs transmission of 1 MB file), do you observe less throughput, why? Explain how you can improve the throughput of that low throughput case and record the observed throughput. Further, comment on how Data Rate of the point-to-point link and observed throughput are related.

Sample code for sending 1MB file instead of echo client-server application

```
NS_LOG_INFO ("Create Applications.");


// Create a BulkSendApplication and install it on node 0 uint16_t
sinkPort = 9; // port number

BulkSendHelper source ("ns3::TcpSocketFactory",
                      InetSocketAddress (interfaces.GetAddress (1), sinkPort));
//TCP sockets are used by BulkSend App to send data to remote node 1 on sinkPort

// Set the amount of data to send in bytes. Zero is unlimited.

source.SetAttribute ("MaxBytes", IntegerValue (1000000)); // 1MB size

ApplicationContainer sourceApps = source.Install (nodes.Get (0)); //install
BulkSend App on node 0
```



```
sourceApps.Start (Seconds (0.0));

sourceApps.Stop (Seconds (10.0));

// Create a PacketSinkApplication and install it on node 1 PacketSinkHelper sink
("ns3::TcpSocketFactory",
    InetSocketAddress (interfaces.GetAddress (1), sinkPort));
//TCP is used to receive pkts on sinkPort of node 1

ApplicationContainer sinkApps = sink.Install (nodes.Get (1)); //install sink app on
node 1

sinkApps.Start (Seconds (0.0));

sinkApps.Stop (Seconds (11.0));
```

[Check Web sources for more information](#)

Task 2: Transport Layer

Goal:

Part 1: Study and understand Transport layer protocol using Wireshark and other tools.

Part 2: Study and understand TCP variants and QUIC by reading research articles.

Part 3: Run network simulations in NS-3 to study variation in Congestion Window.

Part 1: Wireshark/tshark/tcpdump

1. Download a large file (ubuntu image from the **Internet**). 60 seconds of observation is sufficient. Plot the following metrics from Wireshark
 - a. Plot the estimated Round-Trip Time (RTT) variation for download
 - b. Plot the TCP Congestion window (or the difference in ack numbers - bytes delivered) for download. The X-axis is time and Y-axis is bytes delivered (X ticks for each RTT, hence sum up the bytes delivered over each RTT).
 - c. Get the flow graph (Statistics - flow graph)
 - d. What is the average throughput observed?
 - e. Plot the receiver congestion window advertised over time
 - f. Plot the number of 1-duplicate acks, 2-duplicate ack, and 3-duplicate acks received over time
2. Download a small file and identify the TCP 3-way handshake?
3. Ping a host and capture the packets with Wireshark. What kind of packets are generated by the ping command?
4. In case of UDP traffic (skype), make a call and get the pcap
 - a. Plot the interpacket interval (average, max, and min).
 - b. Plot the jitter incurred when skype traffic is used.

Part 2: Study of TCP Congestion Control Algorithms

1. Explain and compare any four TCP congestion control algorithms (TCP Variants).

Out of four TCP congestion control algorithms, one must be TCP Cubic, at least one algorithm should be from loss-based, delay-based, and hybrid categories. Definitions of these categories are given below and some of the TCP variants are shown in the table.

1. Loss-based TCP variants use packet loss as an indication of Congestion.
2. Delay-based TCP variants consider packet delay rather than packet loss as congestion in the network.
3. In the hybrid type of TCP variant, both packet loss and delay of a packet are considered as congestion in the network.

Category	TCP Variants
Loss Based & Loss + Estimation	TCP-Tahoe, TCP-Reno, TCP- New Reno, TCP-SACK, TCP-FACK, , BIC-TCP, TCP Cubic , TCP Hybla TCP, HS-TCP, H-TCP, S-TCP, LP, TCP Libra , TCP Westwood , TCPW CRB, TWPW BR, DOOR,DSACK, TD FR, TCP -FIT
Delay Based	TCP-Vegas , TCP Vegas-A, TCP New Vegas, FAST-TCP, Nice, TCP Real
Hybrid(Loss & Delay) Based & L+D+ Estimation	Compound TCP, , TCP Jersey , TCP Africa, TCP Veno , TCP Illinois', YeAH TCP, TCP Fusion, , TCP-FNC,TCP-ACC

Note: Refer to research papers for different algorithms. The report should have the following points

- ➔ Algorithm Details
- ➔ Suitable for which scenarios and where it fails
- ➔ Compare all four variants in the end and write some inferences

2. Explain the QUIC protocol and its working in detail.
 - ➔ Detailed Working of QUIC and how it is different than other transport layer protocols
 - ➔ Pros and cons of QUIC protocol

References:

1. The QUIC Transport Protocol: Design and Internet-Scale Deployment, Google SIGCOMM 2017.
2. <https://conferences.sigcomm.org/sigcomm/2020/tutorial-quic.html>, QUIC Tutorial, SIGCOMM 2020
3. IETF RFC 9000 QUIC link: <https://datatracker.ietf.org/doc/html/rfc9000>.

Part 3: Congestion Window study using NS-3

Use TCP_Demo.cc file for this part. The file is similar to sixth.cc/seventh.cc files available in examples/tutorial folder of NS-3. In this part of the assignment, you will use two different TCP variants (NewReno and CUBIC) and study the changes in the congestion window over time. You have to run the simulations by varying TCP variants. The TCP variant can be changed as shown below:

```
//TODO: TCP variant set to NewReno/TcpCubic

//Config::SetDefault("ns3::TcpL4Protocol::SocketType", TypeIdValue
(TcpNewReno::GetTypeId()));

Config::SetDefault("ns3::TcpL4Protocol::SocketType", TypeIdValue
(TcpCubic::GetTypeId()));
```

The following code will help you to get the congestion window trace file which is already included in the TCP_Demo.cc

```
//trace cwnd  
  
AsciiTraceHelper asciiTraceHelper;  
  
Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream  
("tcp-demo.cwnd");  
  
ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow",  
MakeBoundCallback (&CwndChange, stream));
```

Use Gnuplot or any other tool to plot the CW values. You can use the following Gnuplot script for the plot.

```
set terminal png  
set output "CW.png"  
set title "Congestion window Plot"  
set xlabel "Time (Seconds)"  
set ylabel "Congestion Window"  
  
plot "tcp-demo.cwnd" using 1:2 with linespoints title "Old Cwnd", "tcp-  
demo.cwnd" using 1:3 with linespoints title "New Cwnd"
```

These lines are copied in the CW.plt which you can run to plot the congestion window over time. After running you will get CW.png.

```
>> gnuplot CW.plt
```

Answer the following questions for both the TCP variants:

1. How many times did the TCP algorithm reduce the cwnd, and why?
2. Out of these two TCP variants which variant is better, and why?

[Check Web sources for more information](#)