COL-334 Computer Networks Assignment-3

Chinmay Mittal (2020CS10336)

Task-1

Description of Code

The code for this part is available in the file *task-1.cc*. This file has been modified from *seventh.cc* which is one of the tutorials provided as part of ns3. The callbacks used in *seventh.cc* for packet loss and change in congestion window size were used for filling the table and making the graphs. `CwndChange` is called whenever the congestion window is changed and this callback is used to keep track of the Maximum congestion window size. This callback also writes the window size and timestamps to an appropriate file e.g *task-1-TcpVeno.cwnd*. This file is then used by the plotting script named *congestion.plt* to make the graph and then save the corresponding graph as an image e.g. *task-1-TcpVeno.png*. `RxDrop` is the callback which is called whenever a packet drop occurs. This callback is used to maintain the count of the packets dropped which is then logged to the console at the end of the script.

```
## running task-1.cc, set the following variable to the TCP congestion
control algorithm required in the main function
## std::string congestionControlAlgorithm = "TcpWestwood" ;
## command to run task-1.cc => ./waf --run scratch/task-1
## change the protocol name in congestion.plt => protocol = "TcpWestwood"
## command to run congestion.plt => gnuplot congestion.plt
## This saves a png with the appropriate name => task-1-TcpWestwood.png
```

The following was used to change the congestion control algorithm used by the TCPSocket

```
 // congestion-algorithm setup
 TypeId tid = TypeId::LookupByName ("ns3::" + congestionControlAlgorithm);
 Config::Set ("/NodeList/*/$ns3::TcpL4Protocol/SocketType", TypeIdValue
(tid));
 Ptr<Socket> ns3TcpSocket = Socket::CreateSocket (nodes.Get (0),
TcpSocketFactory::GetTypeId ());
```

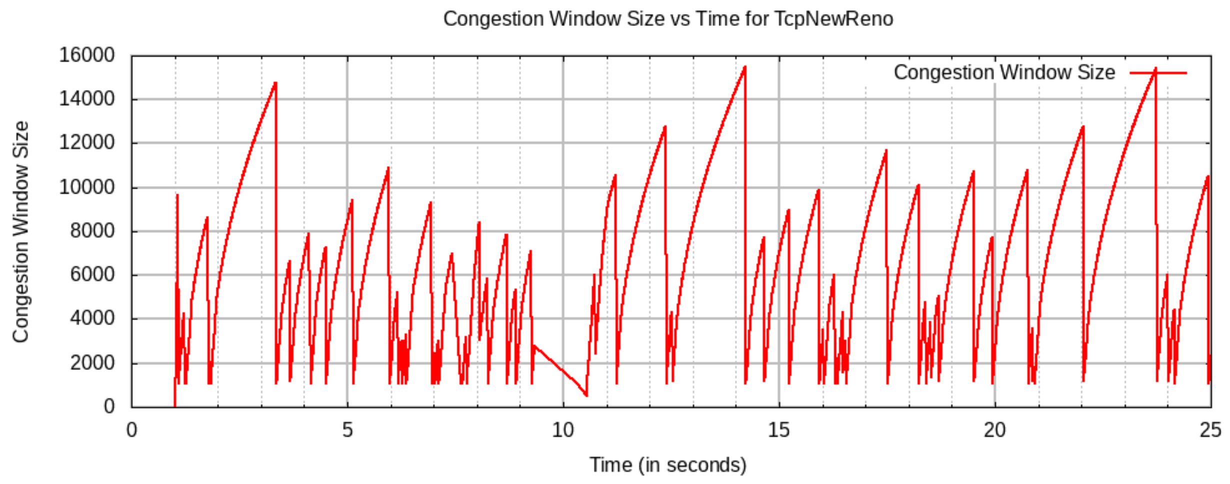
The Maximum congestion window size and the number of packets dropped are logged on the console:

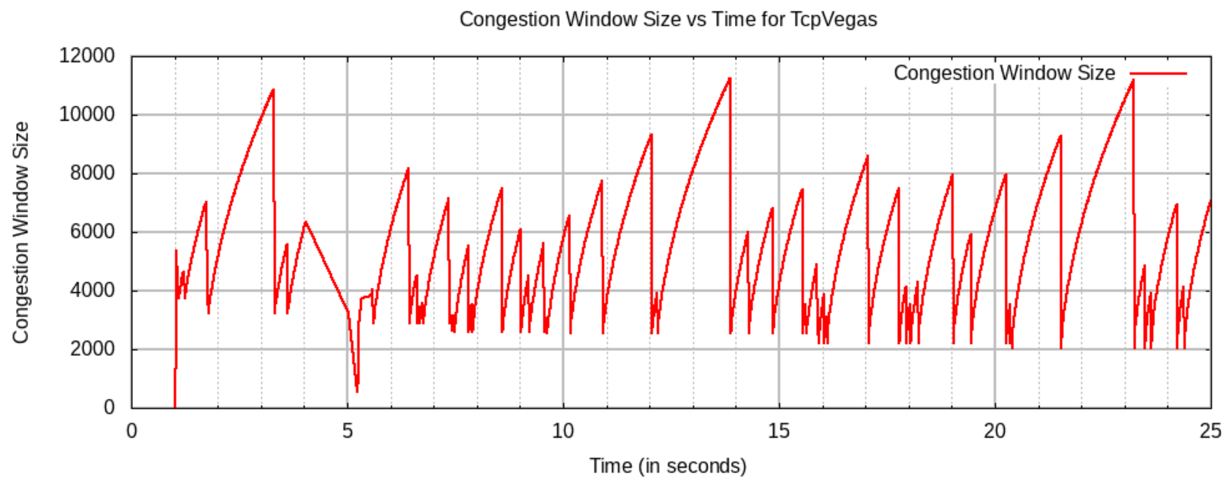
```
Max Window Size: 15471
Number of packets dropped: 60
```

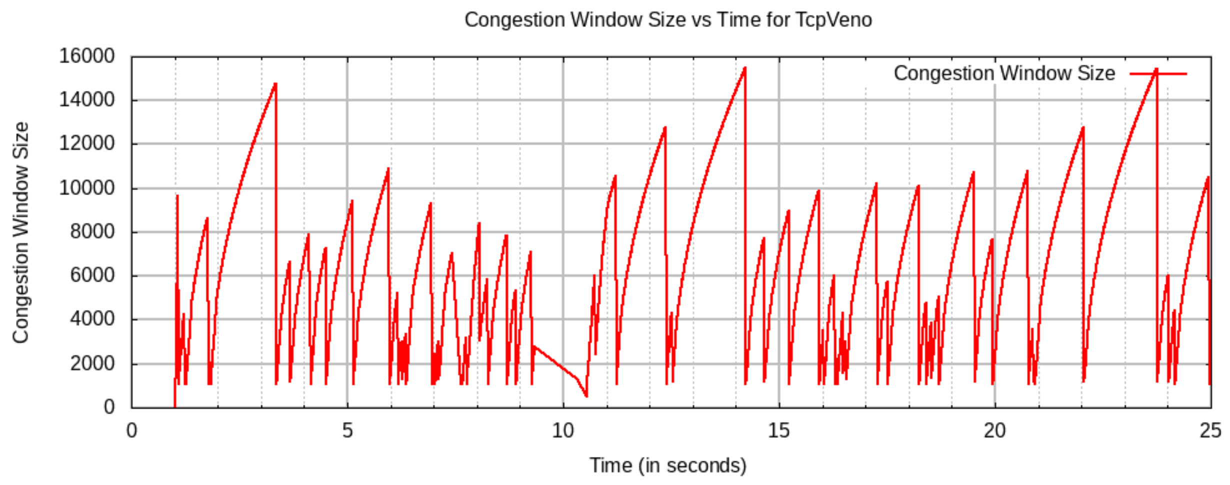
Question-1

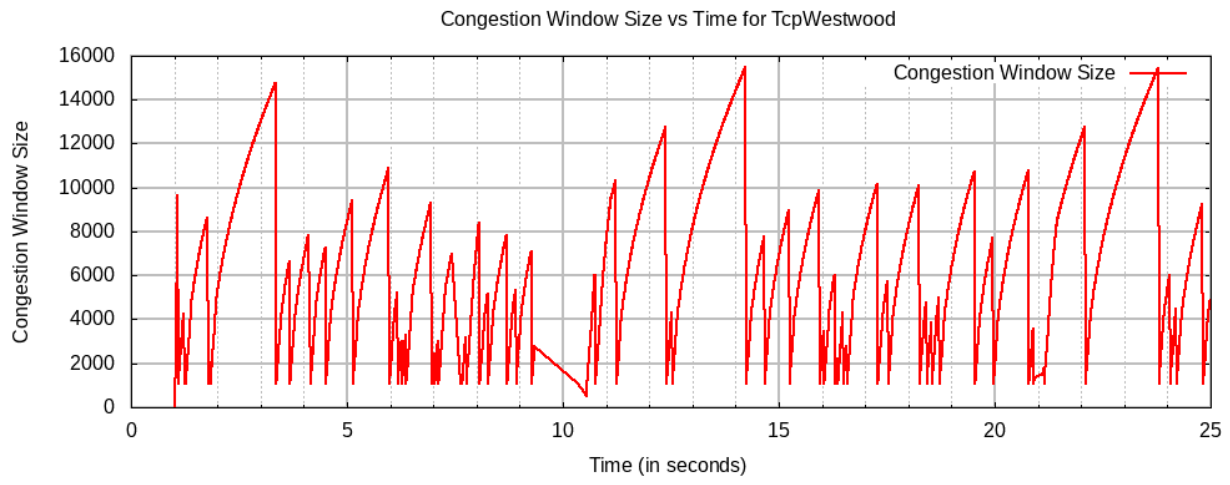
Protocol Name	Max. Window Size	No. of Packet Drop
NewReno	15462	58
Vegas	11252	58
Veno	15462	59
WestWood	15471	60

Question-2









Question-3 (Explanation of Graphs and Congestion control Algorithms)

Due to the RateErrorModel used at the receiver, the sender perceives that the packets are being lost. This causes the sender to perceive congestion which causes it to decrease its congestion window and start increasing it again. This leads to the kind of the saw-tooth behavior as seen in all graphs.

The graphs of TcpNewReno, TcpWestWood and TcpVeno are almost similar to each other whereas that of TcpVegas is different compared to these. NewReno, WestWood and Veno are very similar algorithms and their differences are mainly used to address problems not tested by this simple network hence their graphs almost overlap each other while differing from Veno which is fundamentally different.

TcpVegas is a delay based congestion control algorithm and is fundamentally different from the others it is more passive in its additive phase and tries to prevent congestion thus the Maximum congestion window size is lower compared to others because of this passive nature also the frequency of oscillations is also lower compared to the other algorithms.

TcpNewReno is an improvement of *TcpReno* which uses partial acknowledgments to handle multiple packet losses much better. TcpNewReno does not go out of FastRecovery on receiving partial acknowledgments which prevents the congestion window from being decreased multiple times due to multiple packet loss.

TcpWestWood adapts additive increase and adaptive decrease where instead of halving the congestion window at a loss episode TcpWestWood tries to estimate the network bandwidth and uses this estimate to change the congestion window.

TcpVeno refines the additive increase and multiplicative decrease phase of Reno. It tries to differentiate between congestion loss and random loss, it uses a scheme similar to Vegas where it tries to measure the RTT to estimate the state of the network. If a loss occurs when the estimated state of the network is not in congestion then Veno assumes this loss to be random and only decreases the cwnd to ⅘ of the previous value otherwise if the network state is estimated to be in congestion then it decreases the cwnd to cwnd/2 as before in Reno. Also it improves the additive increase phase where it increases the congestion window slightly slowly once the network is estimated to be in congestion so that it can stay at that state for longer.

TcpVegas is a pure delay-based congestion control algorithm implementing a proactive scheme that tries to prevent packet drops by maintaining a small backlog at the bottleneck queue. It compares the observed throughput with the estimated throughput and tries to bound the difference between the two to prevent buffering. To avoid congestion, Vegas linearly increases/decreases its congestion window to ensure the difference between the observed and expected value falls between the two predefined thresholds, alpha and beta.

Question-4

BBR stands for the Bottleneck Bandwidth and Round-trip propagation time based TCP congestion control algorithm developed by Google in 2016. Most traditional algorithms use loss based triggers for congestions. Since networks have become much more reliable these loss based triggers are no longer as effective as in the old times. BBR uses latency instead of the loss packets as the main factor to determine the sending rate. Most protocols start sending at lower rates when they observe a packet loss which might not indicate congestion but other random factors such as bit errors this significantly decreases the average throughput that these algorithms can achieve when the network has these kinds of random errors. On the other hand BBR tries to estimate the available bandwidth and keep the sending rate close to the available bandwidth thus it does not decrease congestion window abruptly in the face of random errors. Thus BBR achieves much better latency and throughput in such cases.

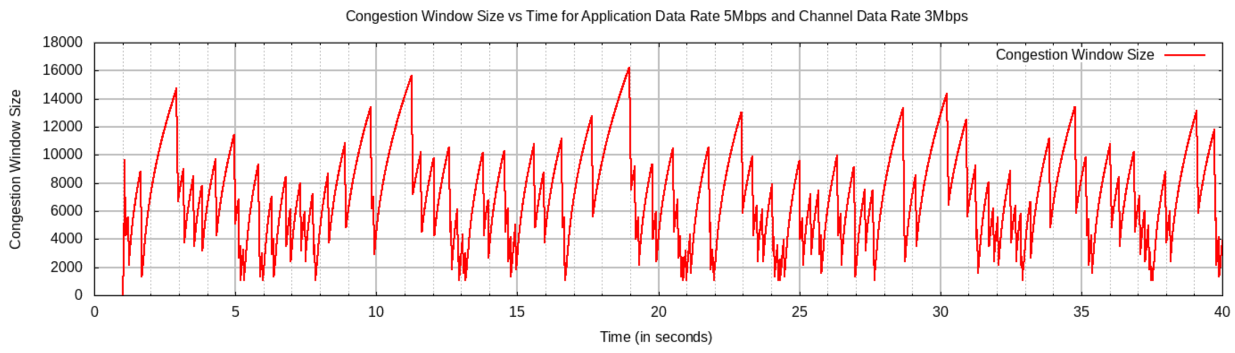
Task-2

Description of Code

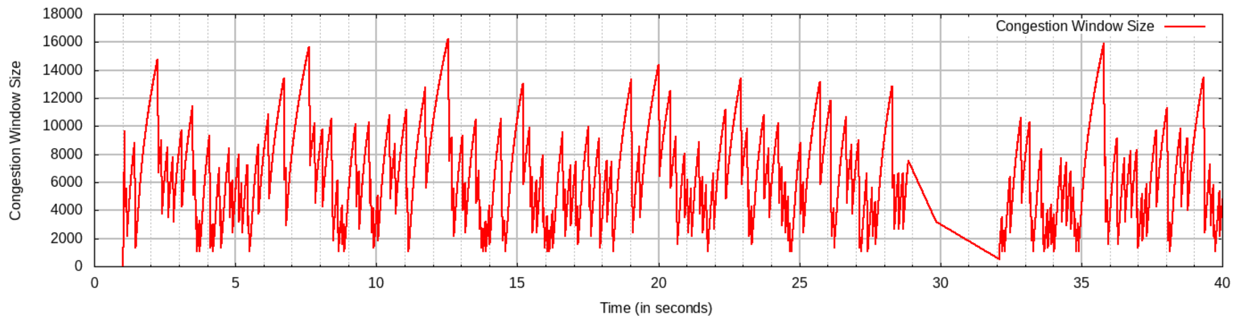
This code is also very similar to that of task-1 and is available in the file task-2.cc. It has also been adapted from ns3's tutorial example file namely seventh.cc. A .cwnd file is generated with the congestion window and the appropriate time stamps e.g. task-2-cdr:10Mbps-adr:5Mbps.cwnd. This file is then used by the plotting script named congestion-2.plt to generate the graph as an image e.g. task-2-cdr-10Mbps-adr-5Mbps.png.

```
## running task-2.cc, set the following variable to the channel data rate
and the application data rate required
## std::string channelDataRate = "4Mbps" ;
## std::string applicationDataRate = "12Mbps" ;
## command to run task-1.cc => ./waf --run scratch/task-2
## change the data rates in congestion-2.plt
## cdr = "4Mbps"
## adr = "12Mbps"
## command to run congestion-2.plt => gnuplot congestion-2.plt
## This saves a png with the appropriate name =>
task-2-cdr-4Mbps-adr-12Mbps.png
```

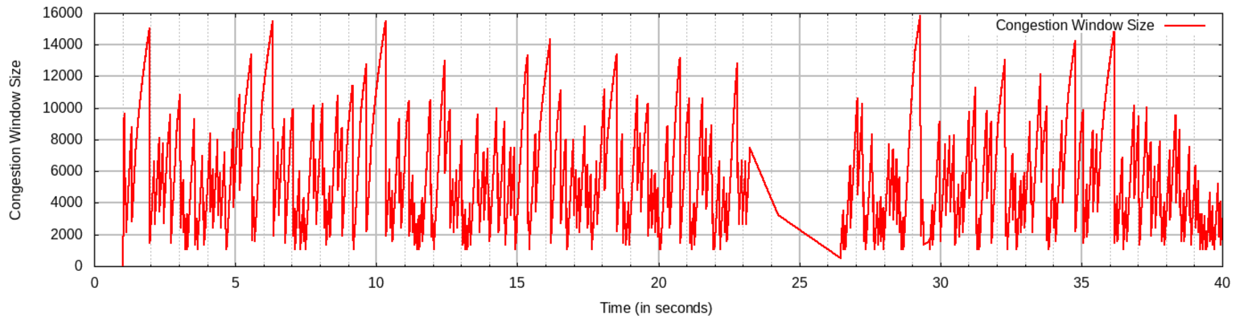
Application Data Rate Constant and Varying Channel Data Rate



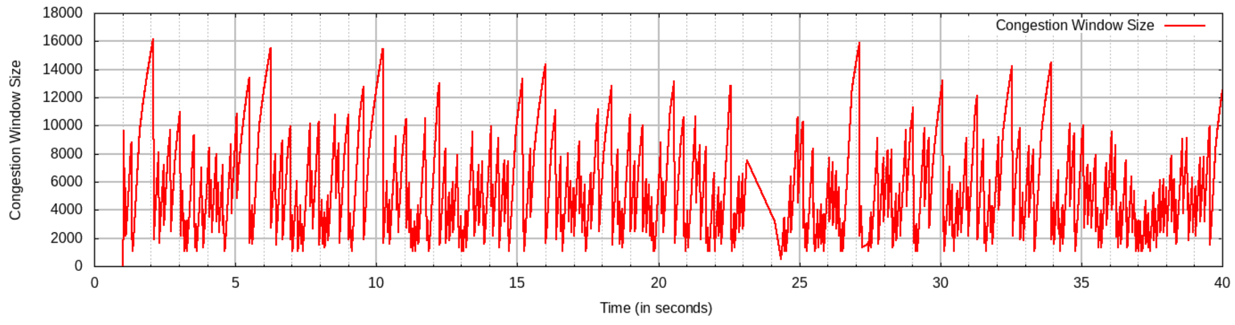
Congestion Window Size vs Time for Application Data Rate 5Mbps and Channel Data Rate 5Mbps



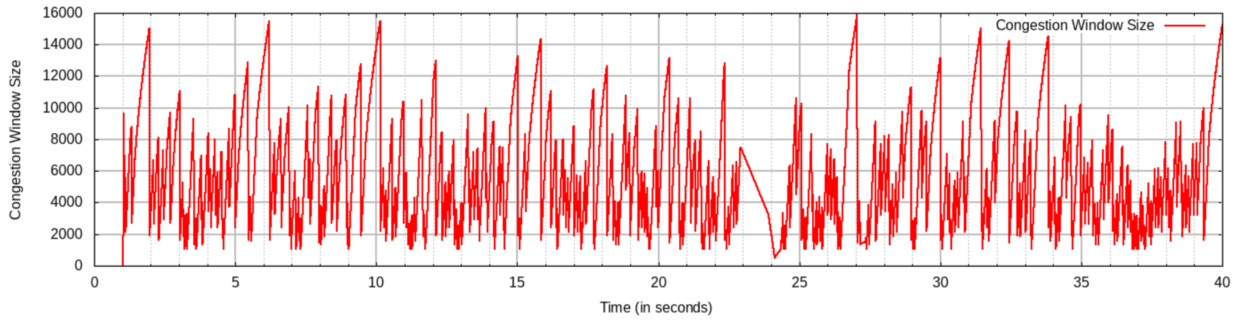
Congestion Window Size vs Time for Application Data Rate 5Mbps and Channel Data Rate 10Mbps



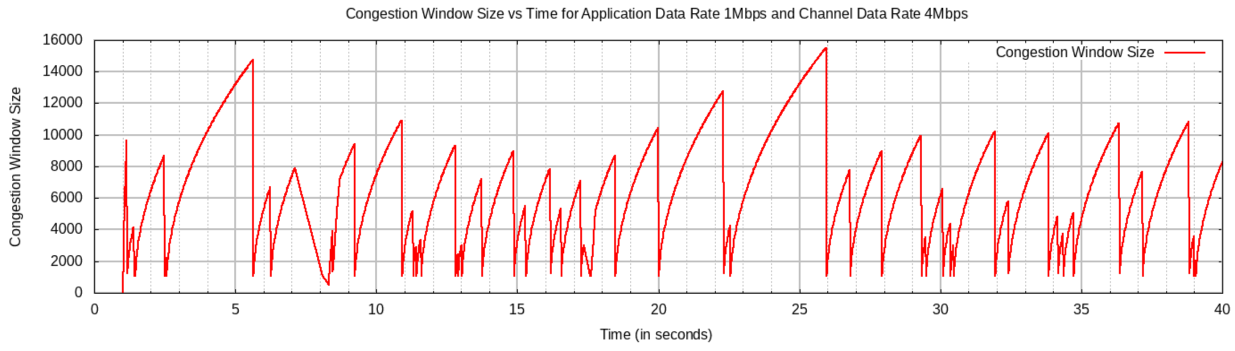
Congestion Window Size vs Time for Application Data Rate 5Mbps and Channel Data Rate 15Mbps

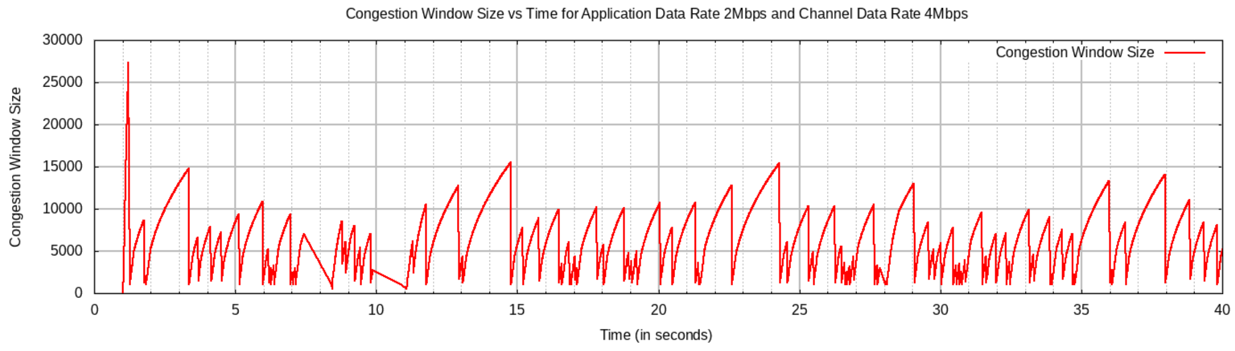


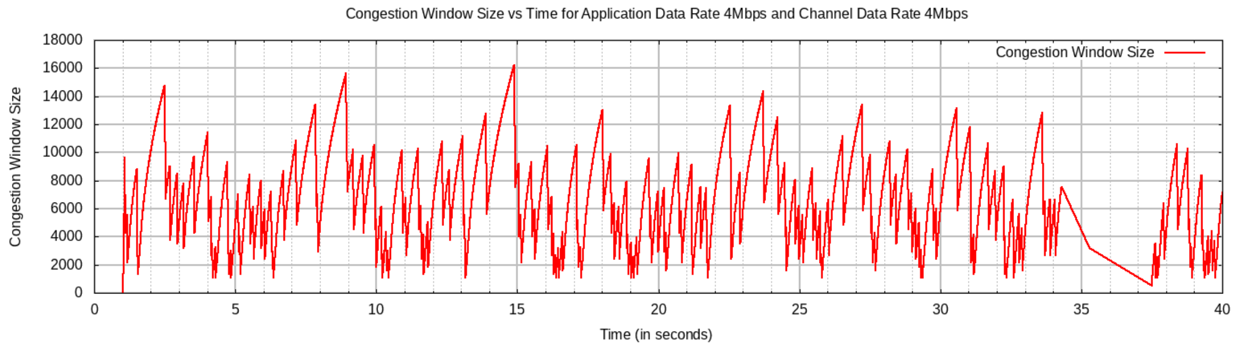
Congestion Window Size vs Time for Application Data Rate 5Mbps and Channel Data Rate 30Mbps

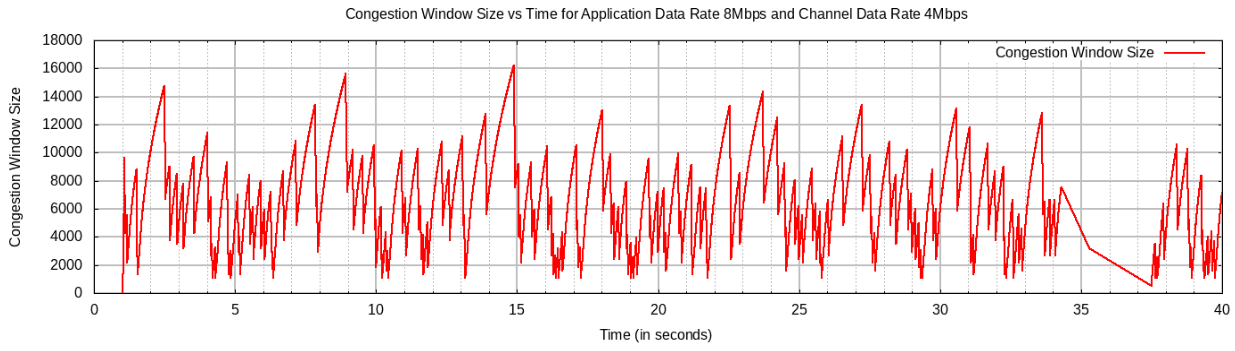


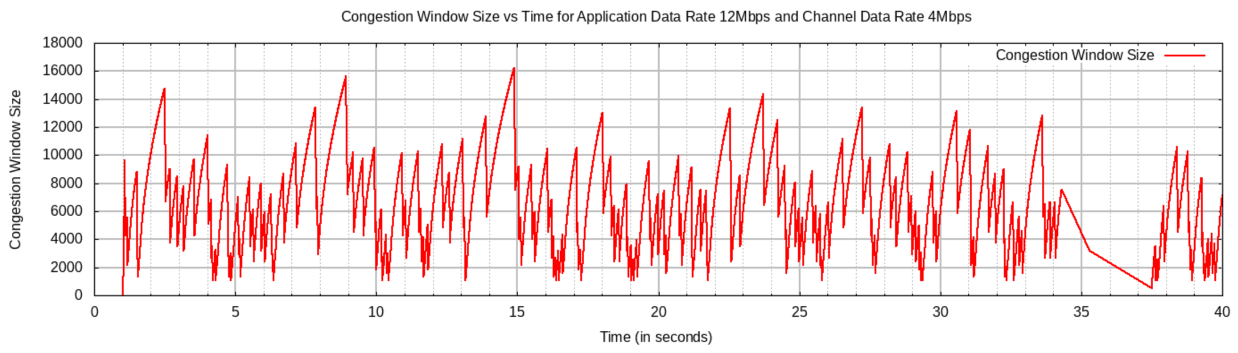
Channel Data Rate Constant and Varying Application Data Rate











How does application data rate affect congestion window

While maintaining the channel data rate as we increase the application data rate, the link will eventually start getting flooded which will lead to more congestion and thus lead to packet loss this will cause additional timeouts leading to more congestion signaling to the sender along with the random packet drops.

This can be seen from the graphs as we increase the application data rate from 1 to 2 and eventually to 4. We can see the frequency of oscillations increasing because the congestion window is decreased more often due to more frequent timeouts.

Once the application data rate increases beyond the channel data rate there isn't much difference in the congestion window size since the extra data available will not be sent because the congestion window will try to avoid overfilling the link, the additional data available will simply be buffered at the sender. Thus the available throughput and thus eventually the congestion/ congestion window size will be bottlenecked by the channel data rate.

How does channel data rate affect congestion window

Keeping the application data rate constant and increasing the channel data rate can help avoid packet queuing at the channel interface thus helping us avoid congestion at the link and helping us achieve much better throughput by allowing a relatively larger congestion window. With smaller channel data rates the application can not achieve the required throughput. We can see from the graph that as the channel data rate increases from 3 Mbps to 5Mbps the graph is essentially squished along the x-axis because the sender can afford to increase the congestion window at a much faster rate.

Relation between the Channel data rate and the Application data rate

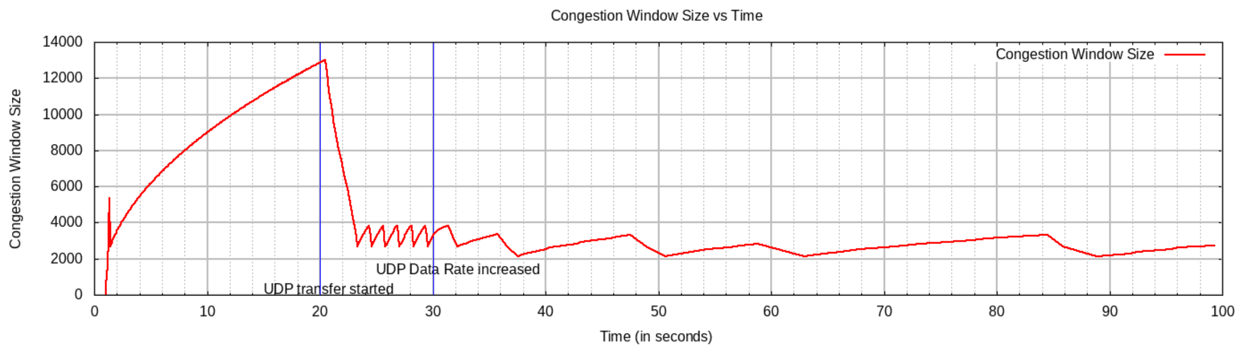
When the application data rate is less than the channel data rate then there is less congestion since there is not much queueing at the buffer, but the channel is also under utilized. When the application data rate is more than the channel data rate then that causes congestion and TCP's congestion control algorithm throttles the sending rate to prevent congestion despite having much more data that is to be sent. It is thus beneficial to keep the channel data rate close to the application data rate in which case the channel is fully utilized without much congestion.

Task-3

Description of Code

To make the multi node point to point network I have followed the following stack overflow post. To design the TCP data transfer I have followed code from the tutorial provided by ns3 from the file seventh.cc. For the UDP data transfer I have referred to the following tutorial provided by ns3. As before a file named task-3.cwnd is created with the congestion window sizes and the timestamps which is used by the plotting script named congestion-3.plt to generate the graph as an image named *task-3.png.*

```
## running task-3.cc,
## command to run task-3.cc => ./waf --run scratch/task-3
## command to run congestion-3.plt => gnuplot congestion-3.plt
## This saves a png with the appropriate name => task-3.png
```



Observation

Before 20s, node-1 (TCP source) keeps increasing its congestion window to probe the network capacity. Since the channel data rates are more than the application data rates and there is no rate error model congestion does not occur as seen by continuously increasing congestion window.

As soon as the UDP source (node-2) starts sending data at 20s, this floods the link connecting node-2 and node-3. For this link the incoming data rate reaches very close to the channel data rate and thus causes queuing and packet drops. This causes congestion at the TCP source (detected by the packet drops/timeouts due to queueing) which decreases its congestion window drastically. The TCP source keeps probing the network by increasing its congestion window and as soon as congestion starts again it decreases its congestion window, this keeps happening periodically according to the specifications of TCPVegas.

At 30s the UDP source increases its data rate further which causes further congestion at the common link. This causes the observed RTTs by node-1 to increase because of more queueing. This changes the time-period of the oscillation because of the way TCPVegas works. TCP Vegas's congestion detection scheme checks every RTT whether network conditions have changed enough to evoke a change in the congestion window adjustment policy. Since RTT increases this happens less frequently which means that the TCP source increases/decreases its congestion window at a slower rate leading to an increase in the time-period of the oscillation after 30s. Also the peak congestion window size after 30s is a little less due to more congestion in the network.

Files Submitted

Note to run the cc files put them in the scratch folder inside installed ns3 distribution

- task-1.cc => code file for Task-1 (change protocol name in code)
- task-2.cc => code file for Task-2 (change data rates in code)
- task-3.cc => code file for Task-3
- congestion.plt => plotting file for Task-1 (change protocol name in the file)
- congestion-2.plt => plotting file for Task-2 (change data rates in the file)
- congestion-3.plt
- 2020CS10336_report.pdf (report containing observations and graphs)
- 8 pcap files => task-3-n1-0.pcap task-3-n2-0.pcap task-3-n2-1.pcap task-3-n3-0.pcap task-3-n3-1.pcap task-3-n3-2.pcap task-3-n4-0.pcap task-3-n5-0.pcap. One for each interface in Task-3