ECE 6110 Lab Assignment 2: Congestion Control

In this lab, you will create a topology, as specified below, and compare and contrast the performances of the TCP CUBIC and NewReno congestion control algorithms. This lab builds on the discussions in class of the TCP CUBIC paper, which can be found at:

https://blough.ece.gatech.edu/6110/tcp_cubic_paper.pdf

You can find example code for most of what is needed for this assignment in the following program:

$NS_DIR/examples/tcp/tcp-variants-comparison.cc

where $NS_DIR stands for the top-level directory in which you installed ns-3. The fifth.cc program from the ns-3 tutorial also introduces the concept of congestion window tracing, although it uses a different sending application from this assignment, which has a different method for tracing the congestion window changes. Therefore, I recommend the tcp-variants-comparison.cc program as a starting point. However, to produce results matching the instructor's code, you should remove the following lines from tcp-variants-comparison.cc:

```
Config::SetDefault ("ns3::TcpSocket::RcvBufSize", UintegerValue (1 << 21));
Config::SetDefault ("ns3::TcpSocket::SndBufSize", UintegerValue (1 << 21));
Config::SetDefault ("ns3::TcpSocketBase::Sack", BooleanValue (sack));
Config::SetDefault ("ns3::TcpL4Protocol::RecoveryType",
                    TypeIdValue (TypeId::LookupByName (recovery)));
```

and use the error model in fifth.cc:

```
Ptr<RateErrorModel> em = CreateObject<RateErrorModel> ();
em->SetAttribute ("ErrorRate", DoubleValue (errorRate));
```

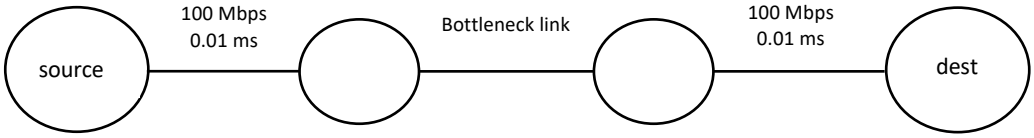
instead of the one in tcp-variants.cc. The model above defaults to byte-level errors, while the one in tcp-variants sets the error unit to packets.

Turn-in Instructions: Create a separate folder for each part of the assignment inside a main folder named "Lab2_<your_last_name>_<your_first_name>". Then create a single tarball of the main folder and all of its subfolders and submit it in Canvas.

Important programming instructions: All code must be hand typed in your editor of choice. No auto-formatting is allowed to change the text that you type in manually. Also, make sure to use the exact program name and command-line parameter names specified in each part. You will get points deducted if you do not follow these instructions even if your code is functionally correct!!

Part 1: Topology with a Bottleneck Link and Homogeneous Flows

Using what you learned from Lab 1, create the depicted topology with 4 nodes and 3 point-to-point links.



Set the data rates and delays on the link from the source and the link to the destination as shown. Make the data rate, delay, and error rate on the bottleneck link command-line parameters in your program. Also include command-line parameters for the number of flows that will be sent from source to destination (maximum of 20) and the protocol to use (TcpCubic or TcpNewReno). Name your program "lab2-

part1.cc" and name the command line parameters "dataRate", "delay", "errorRate", "nFlows", and "transport_prot". Using the BulkSend application from tcp-variants-comparison.cc, set up nFlows TCP flows from the source node to the destination node (set the parameter data_mbytes to 0 for each flow to make the flow send data as fast as possible for the duration of the simulation). The bottleneck link in this topology represents the Internet. Set the default parameters for the bottleneck link to be 1 Mbps data rate, 20 msec delay, and an error rate of 0.00001. Your program should output congestion window changes during the simulation and the goodput achieved by each flow at the end of the simulation.[1] Set the simulation duration to 20 seconds as is done in fifth.cc, start the sink applications at time 0, and start all the flows at time 1 second.

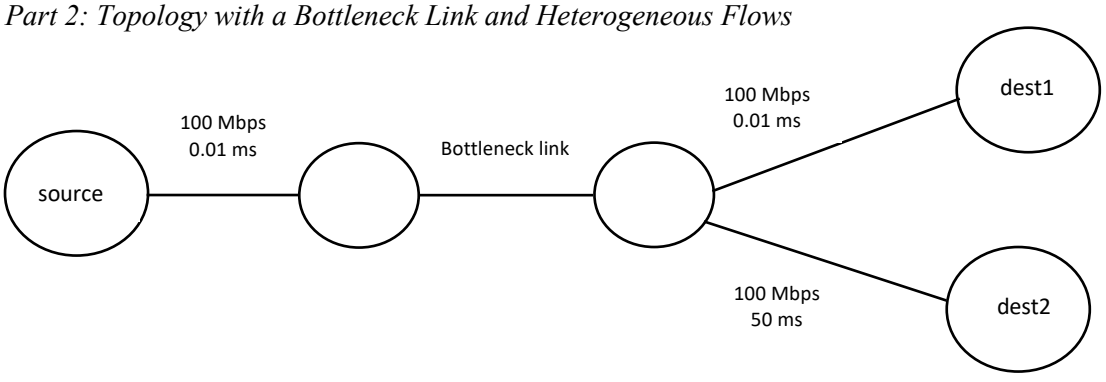
Part1a: For a bottleneck link data rate of 10 Mbps, delay of 100 msec, and error rate of 0.00001, and with one flow, plot the congestion window size of TCP with CUBIC and TCP with NewReno. Compare and contrast these two plots. Also, comment on the shape of the CUBIC plot. Do you see some behaviors that were discussed in class? If so, mention what those are and point them out in the plot. Also, report the goodputs achieved by CUBIC and NewReno in this example and compare them using what you learned from the CUBIC paper.

Part1b: For a bottleneck link data rate of 1 Mbps and error rate of 0.00001, and for 1, 2, and 4 flows, plot the aggregate goodput across all flows vs. bottleneck link delay for delays of 50 to 300 msec in steps of 50 and for both TCP CUBIC and TCP NewReno. Compare and contrast the CUBIC and NewReno goodput behaviors. (There should be 6 line plots on your graph; aggregate goodput for 1, 2, and 4 flows for CUBIC and the same for NewReno.)

Part1c: For a bottleneck link data rate of 1 Mbps and delay of 1 msec, and for 1, 2, and 4 flows, plot the aggregate goodput across all flows vs. bottleneck link error rate for error rates of 0.00001, 0.00005, 0.0001, 0.0005, and 0.001 for both TCP CUBIC and TCP NewReno. Compare and contrast the CUBIC and NewReno goodput behaviors. (There should be 6 line plots on your graph; aggregate goodput for 1, 2, and 4 flows for CUBIC and the same for NewReno.)

Turn-in instructions: In your Part 1 subfolder, include the source code of your program with the given topology, all plots specified above, and your discussion about each set of results as called for above. Also, include screen shots and text files of two sample outputs, one for CUBIC and one for NewReno, with 4 flows and the link data rate, delay, and error rate values from Part 1a.

Part 2: Topology with a Bottleneck Link and Heterogeneous Flows



In this part, the nFlows TCP flows you create will be split equally across the two destinations, which have very different delays on their last links. (Assume the number of flows is always an even number.) This will result in the RTTs of the two sets of flows being quite different. Name your program "lab2-part2.cc"

[1] Goodput refers to the amount of data received at the packet sink application divided by the time that the flow was active (19 seconds given the start and stop times specified above). Report the goodput in bits per second.

and have the same command-line parameters as in Part 1. Run the following experiments and plot the results. For nFlows = 2, 4, 6, and 8, and for a bottleneck link data rate of 1 Mbps, a bottleneck link delay of 20 msec, and an error rate of 0.00001, calculate the average goodput of the flows to dest1 and the average goodput of flows to dest2 for both TCP CUBIC and TCP NewReno. To ensure you get representative behavior, run this experiment 10 times for each value of nFlows and each algorithm and average the results across all runs. Here, you should generate a different sequence of random numbers for each run.[2] Your graph should show average goodputs for dest1 and dest2 flows versus nFlows for the values of nFlows specified and for both CUBIC and NewReno (4 line plots with 4 data points each). Comment on the RTT fairness of the two algorithms and how the results compare to the goodputs you observed in Part 1. All parameters not specified here should be the same as in Part 1.

Turn-in instructions: In your Part 2 subfolder, include the source code of your program with the modified topology, the goodputs vs. nFlows graph, and your discussion about goodputs and RTT fairness. Also, include screen shots and text files of two sample outputs, one for CUBIC and one for NewReno, with 4 flows, the default RNG seed (123456789), and other parameters as specified above.

[2] You can either do this by having a different random number seed each time you run your program, e.g. with the following line of code at the beginning of your program: RngSeedManager::SetSeed(time(NULL));
or you can advance the run number after each run using RngSeedManager::SetRun(), as described in the ns-3 documentation on "Random Variables".