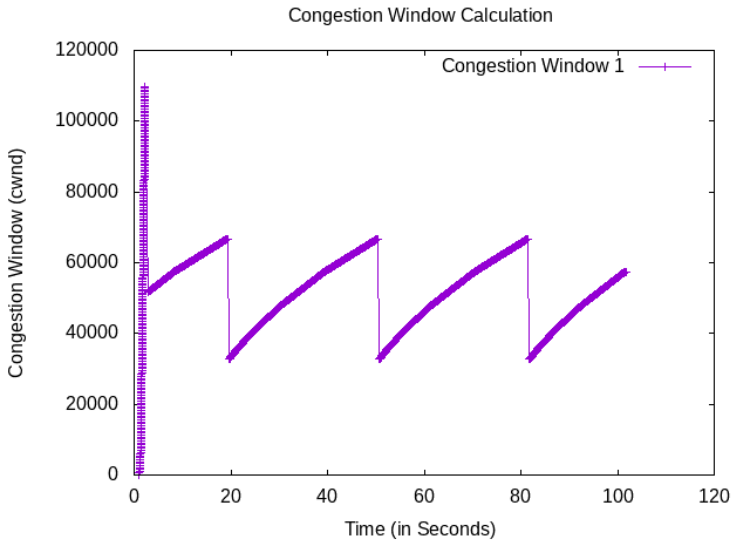
20170058 Keonwoo Kim · November 9, 2020
CS341 Introduction to Computer Networks

Practice #3

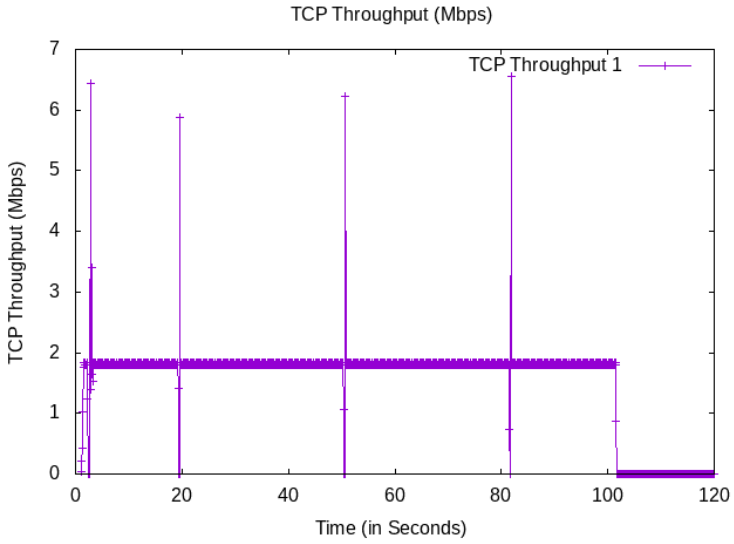
Task 1

1.1. Scenario 1: When app (a) is running (skeleton code).

1.1.1. Plot the *cwnd* (congestion window) of app (a) on a graph.

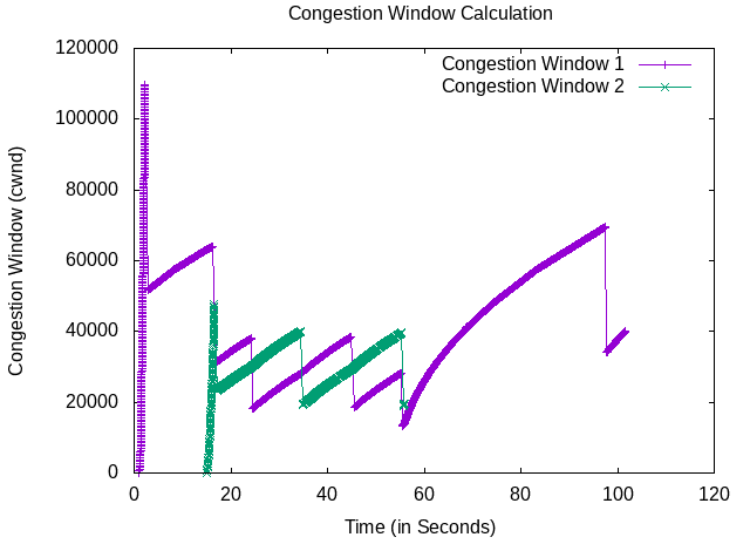


1.1.2. Plot the TCP throughput of app (a) in each 0.1 second. (unit x:sec, y:Mbps).

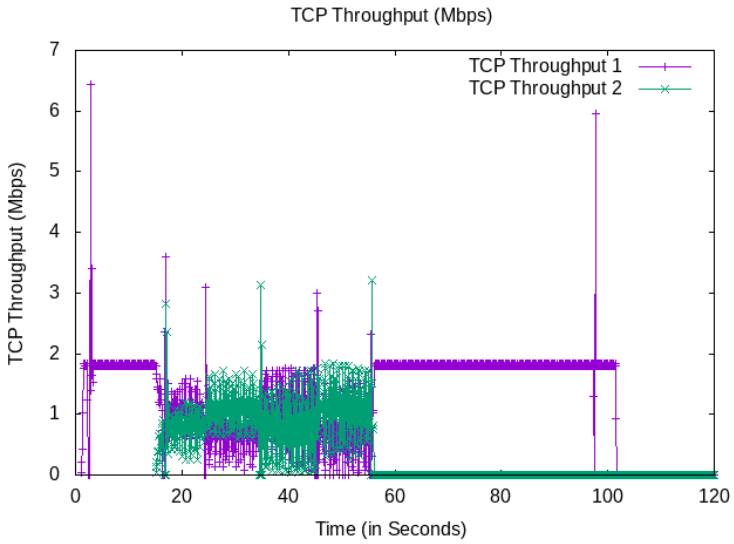


1.2. Scenario 2: When app (a, b) are running.

1.2.1. Plot *cwnd* of app (a, b) respectively on a graph.



1.2.2. Plot the calculated TCP throughput of each app (a, b) in each 0.1 second. (unit x:sec, y:Mbps).

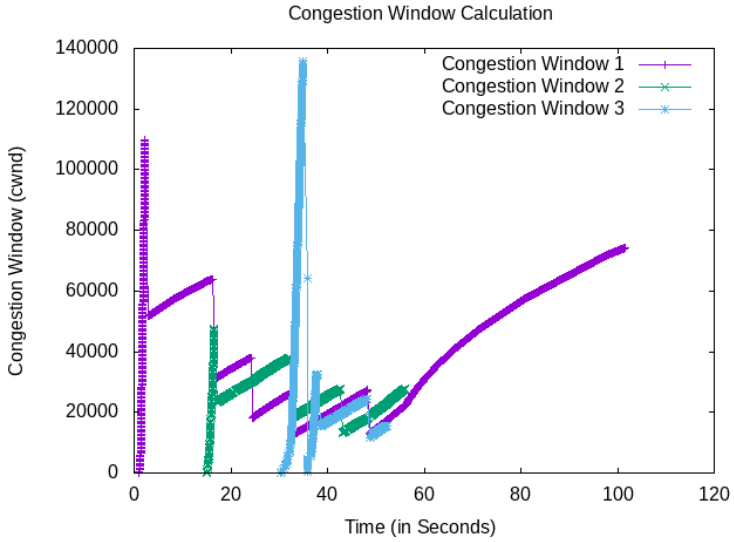


1.2.3. Describe how the graph is different from **Scenario 1 (1.1.)** and discuss the reason why.

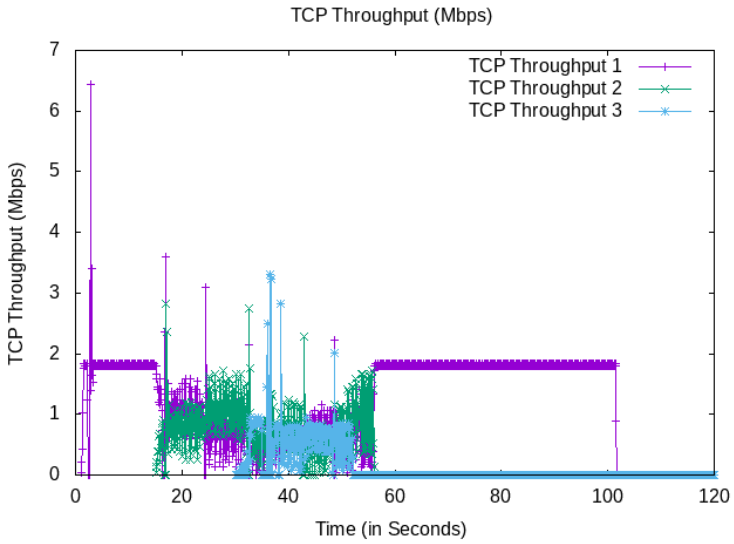
As there are one more application using the same link (from node 1 to node 0,) packet losses occur more frequently so the cwnd's have decreased for app (a) in comparison to one in 1.1.1. In the throughput graph, one can see that two apps are competitively sending data via the shared link.

1.3. Scenario 3: When app (a, b, c) are running.

1.3.1. Plot *cwnd* of app (a, b, c) respectively on a graph.



1.3.2. Plot the calculated TCP throughput of each app (a, b, c) in each 0.1 second. (unit x:sec, y:Mbps).

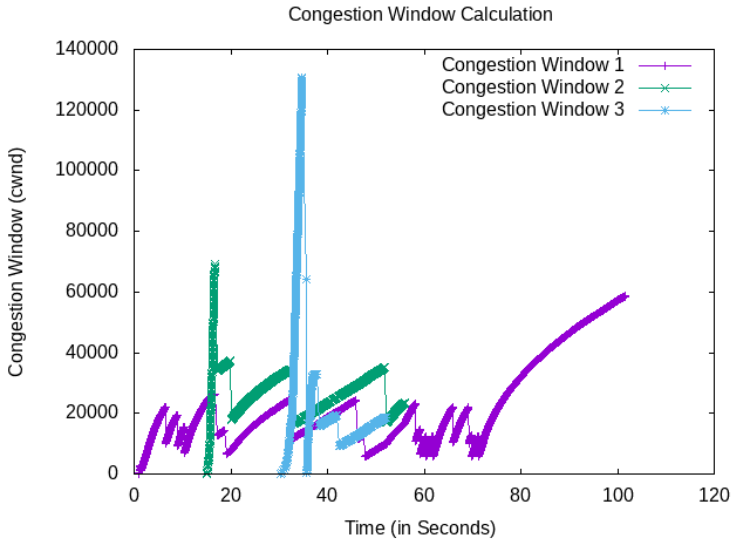


1.3.3. Describe how the graph is different from **Scenario 2 (1.2.)** and discuss the reason why.

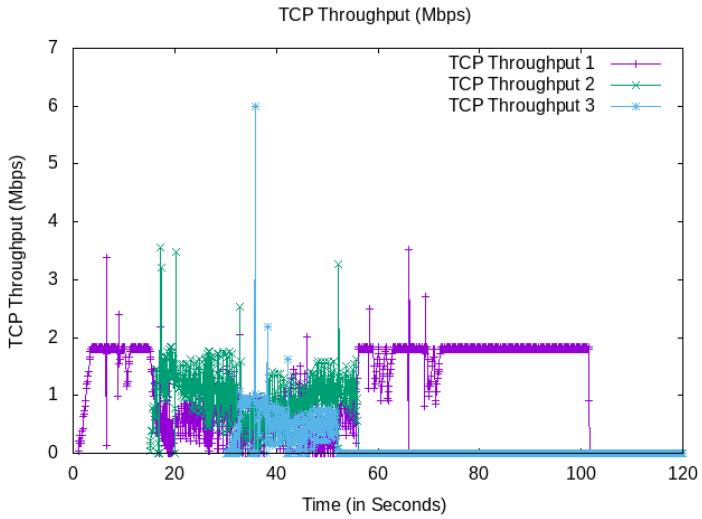
The reason is basically the same with one of 1.2.3., but there is one more app running from $t = 30$ (s). Three apps are sending packets through the same router and the same link to the node 0, so the packet losses became more frequent and hence the level of cwnd again decreased compared to the scenario 2.

1.4. Scenario 4: In real world scenario, there can be corrupted packets when the sink app receives the packets. On the top of Scenario 3, set the receive error rate of 1e-6 set from Node 1 to Node 0.

1.4.1. Plot *cwnd* of app (a, b, c) respectively on a graph.



1.4.2. Plot the calculated TCP throughput of each app (a, b, c) in each 0.1 second. (unit x:sec, y:Mbps).



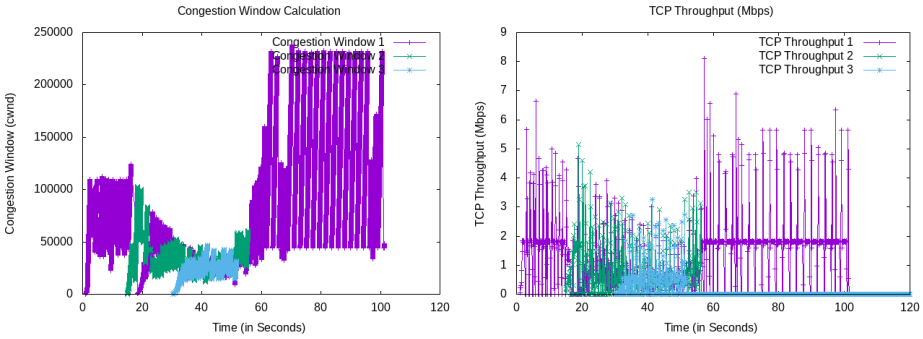
1.4.3. Describe how the graph is different from **Scenario 3 (1.3.)** and discuss the reason why.

Now, packet losses occurs not only due to burst router but also by the receiver. Packet losses are again increasingly occurring, so we can see there are much more TCP recoveries presented.

Task 2

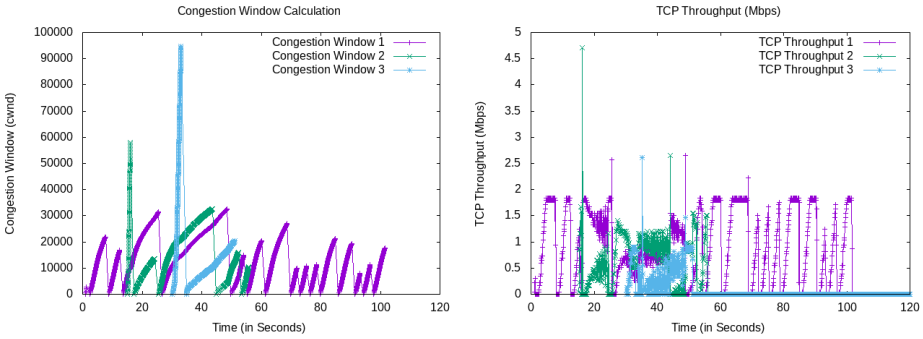
2.1. The default TCP congestion control algorithm in NS3 is set to *TcpNewReno*, and this runs based on the `src/internet/model/tcp-congestion-ops.cc` and `src/internet/model/tcp-recovery-ops.cc`.

2.1.1. In `tcp-congestion-ops.cc`, set the adder in *CongestionAvoidance* function as Appendix 1. Plot *cwnd* and TCP throughput graphs. Describe how the graph is different from **Scenario 4 (1.4.)** and discuss the reason why.



With the original code, we have $\Delta cwnd$ (or $d(cwnd)/dt$) $\propto 1/cwnd$ so that $cwnd(t) \propto \sqrt{t}$, this is the reason why we got a bent curve in cwnd graphs above. When we change that adder to make $\Delta cwnd \propto 1$, we have $cwnd(t) \propto t$, which means it is linear to time, with a rapid rate as the graph shows. Therefore *cwnd* repeats to grow very rapidly and to drop due to the recovery algorithm.

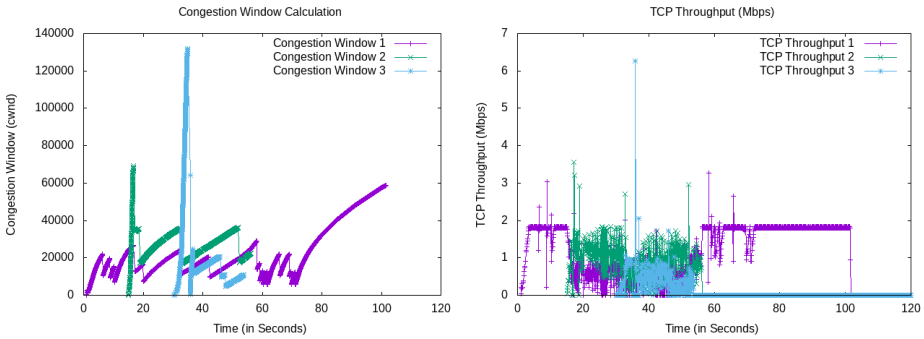
2.1.2. Roll back the changes in 2.1.1. to the original. Now in `tcp-recovery-ops.cc`, set the *EnterRecover*, *DoRecovery*, *ExitRecovery* functions to be as Appendix 2 so that there is no recovery operation. Plot cwnd and throughput graphs. Describe how the graph is different from **Scenario 4 (1.4.)** and discuss the reason why.



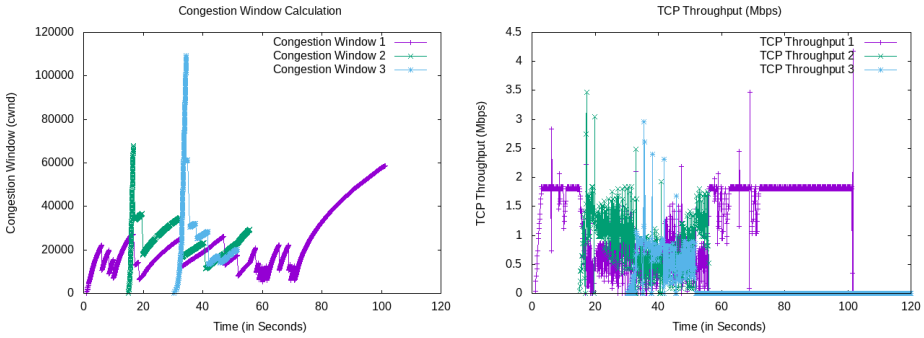
The modification in Appendix 2 makes *cwnd* always be 1 after the recovery. (cWndInfl exists only for the backward compatibility.) We can see the effect of this modification easily in the cwnd graph. (Every recovery in cwnd graph starts at the very bottom, which means $cwnd = 1$.)

2.1. Roll back the previous changes in 2.1. to the original. Try **Scenario 4 (1.4)** using another TCP congestion control algorithms in NS3.

2.2.1. Plot *cwnd* and *TCP* throughput graphs when the algorithm is set to *Veno*.



2.2.2. Plot *cwnd* and *TCP* throughput graphs when the algorithm is set to *Yeah*.



2.2.3. Record the total received bytes from the sink app of (a, b, c) respectively of the **Scenario 4** simulation on each TCP congestion algorithm: *NewReno (NS3 default)*, *Veno*, and *Yeah*. Which algorithm do you prefer and why? Support your answer with evidence such as TCP throughput, packet loss, and congestion.

Using flow monitor helper of ns3, we can observe the total received bytes for each app.

	NewReno	*Veno*	*Yeah*
App (a)	17328600	17526020	17721660
App (b)	5437312	5380268	4921760
App (c)	1535180	1430460	1784876

Table 1: Table of total received bytes

Among three, I think Yeah performs the best, as it recovers cwnd well when the network experiences a packet loss, and the total received bytes for the apps (a) and (c) are way better than other algorithms.