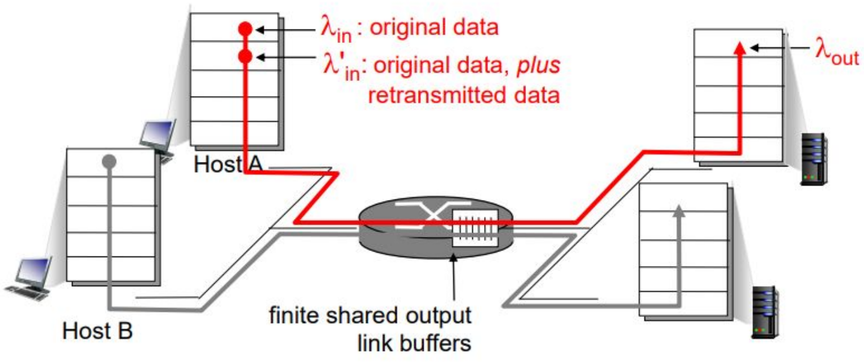
Practice #3: TCP Congestion Control

Sumin Han (hsm6911@kaist.ac.kr), Kisoo Kim (k014520@kaist.ac.kr)

Due: 11/9 Mon, 11:59 pm.

Goal

- Understand why we need TCP congestion algorithm, and how the network can handle the congestion situations.
- In this task, you are required to use **NS-3** for network simulation.



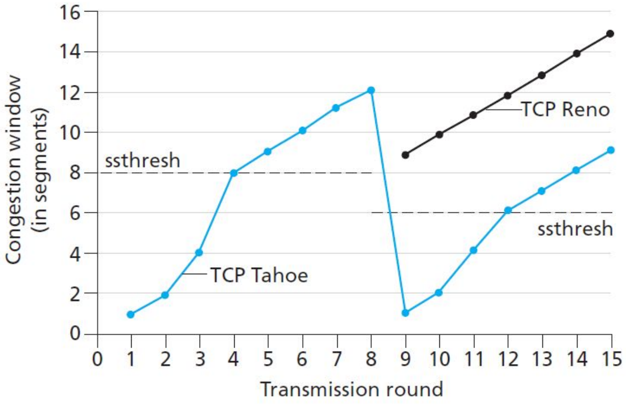
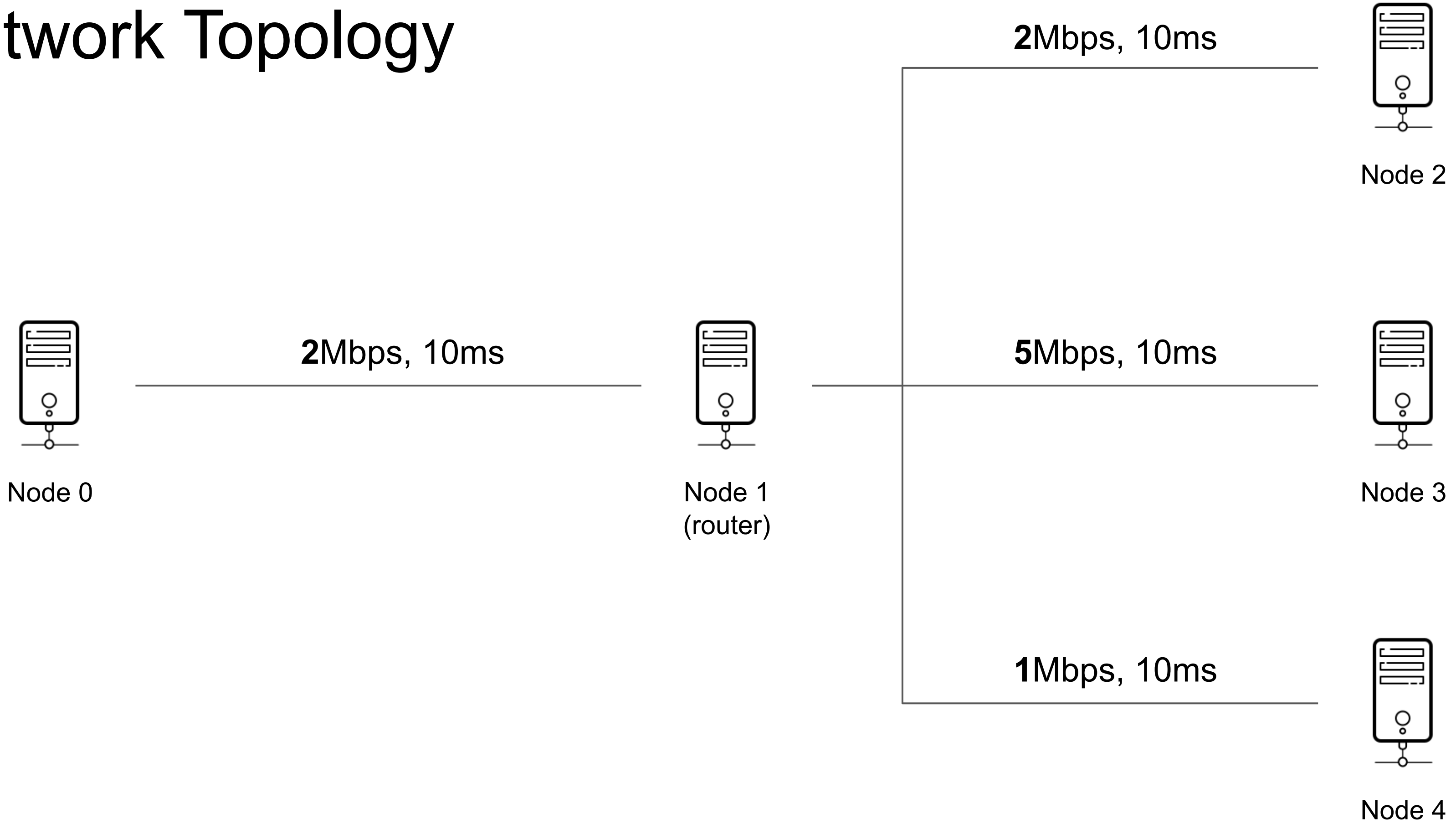


Figure 3.53 ♦ Evolution of TCP's congestion window (Tahoe and Reno)

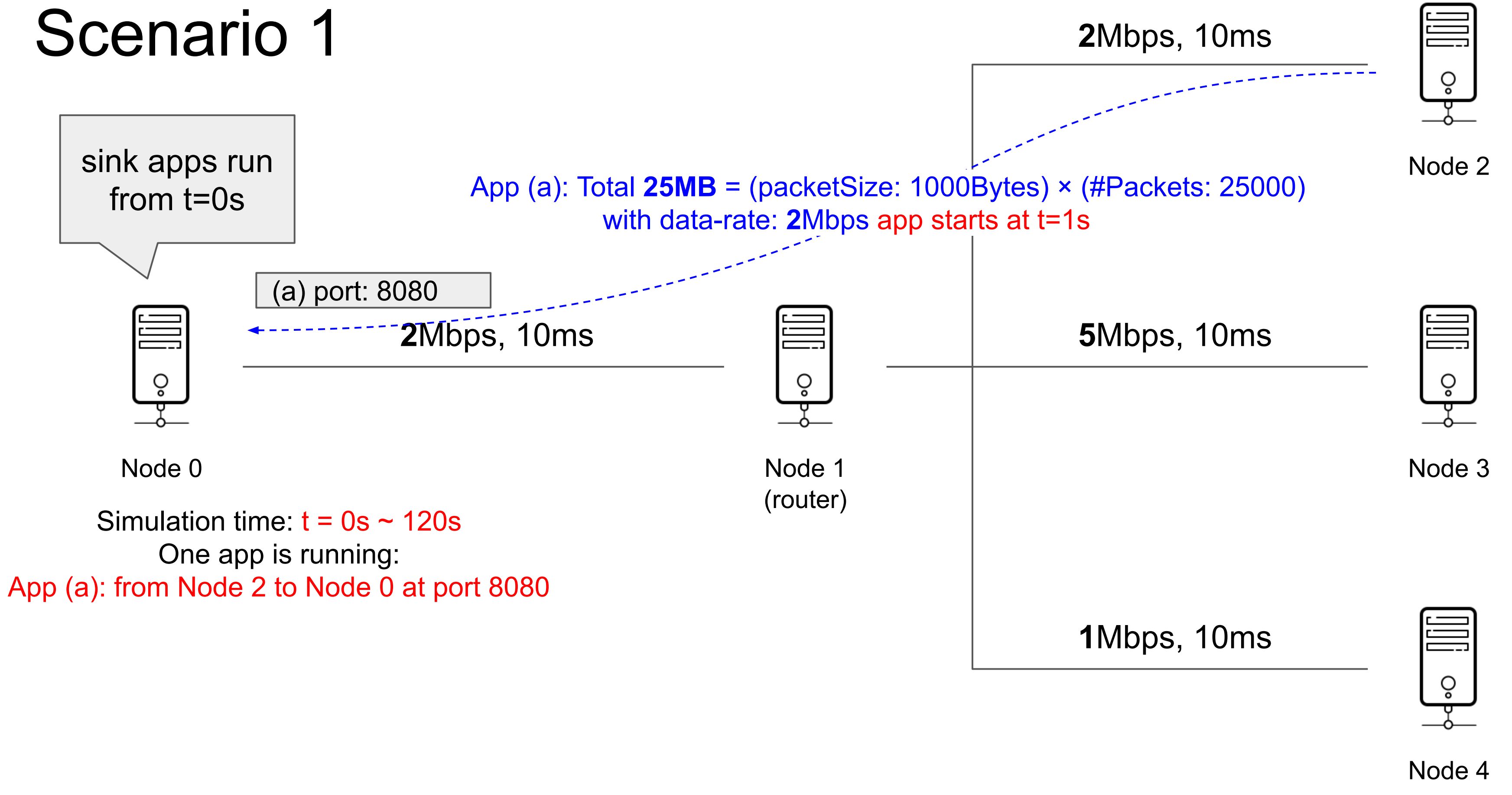
Background

Network congestion may occur when a sender overflows the network with too many packets. At the time of congestion, the network cannot handle this traffic properly, which results in a degraded quality of service (QoS). The typical symptoms of a congestion are: excessive packet delay, packet loss and retransmission. (reference: https://www.noction.com/blog/tcp-transmission-control-protocol-congestion-control)

Network Topology

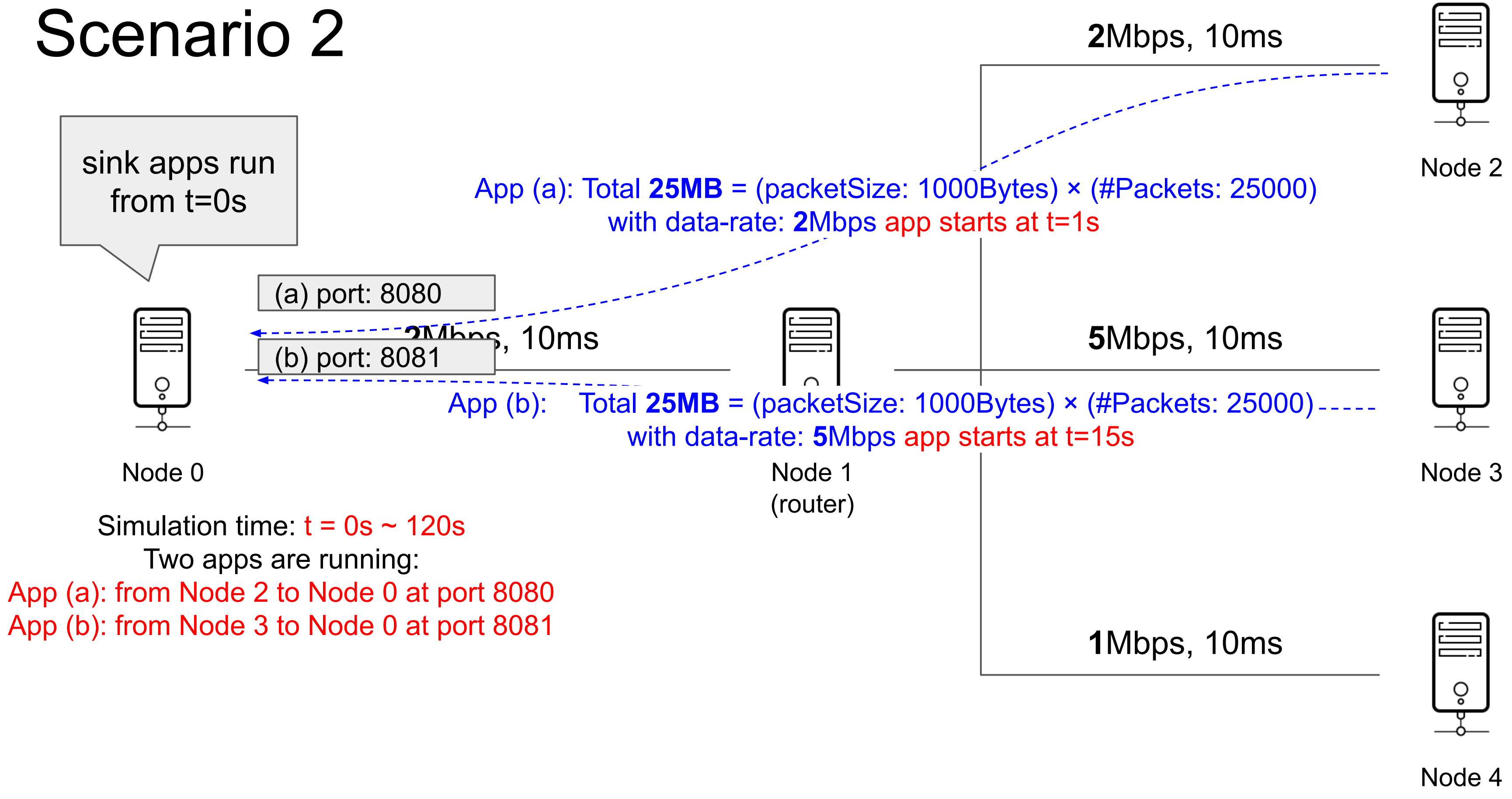


Scenario 1



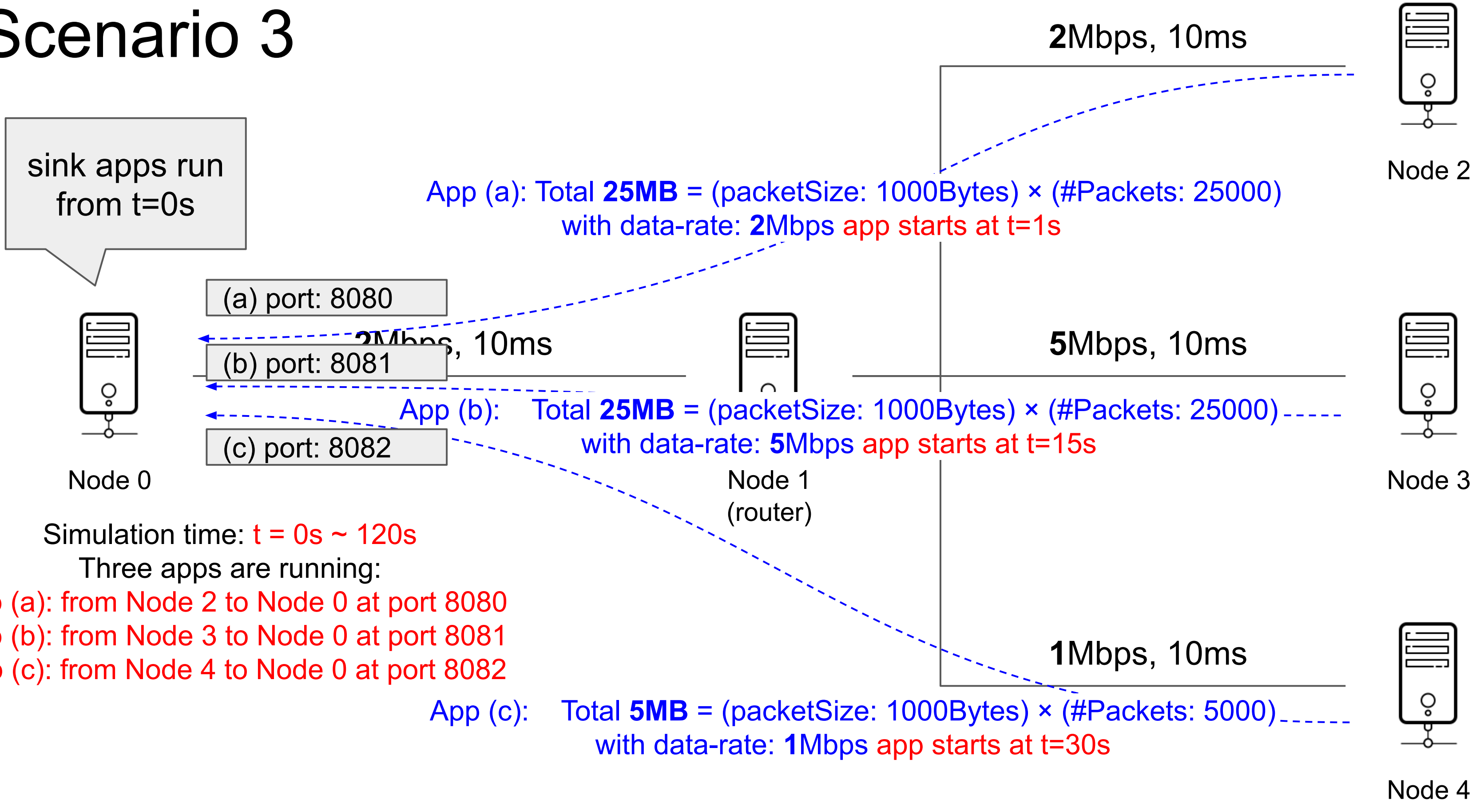
sink apps run from t=0s
(a) port: 8080
2Mbps, 10ms
5Mbps, 10ms
1Mbps, 10ms
App (a): Total **25MB** = (packetSize: 1000Bytes) × (#Packets: 25000)
with data-rate: **2**Mbps app starts at t=1s
Node 0
Node 1 (router)
Node 2
Node 3
Node 4
Simulation time: t = 0s ~ 120s
One app is running:
App (a): from Node 2 to Node 0 at port 8080

Scenario 2



sink apps run
from t=0s
(a) port: 8080
(b) port: 8081
2Mbps, 10ms
Node 0
Node 1
(router)
2Mbps, 10ms
5Mbps, 10ms
1Mbps, 10ms
Node 2
Node 3
Node 4
App (a): Total 25MB = (packetSize: 1000Bytes) × (#Packets: 25000)
with data-rate: 2Mbps app starts at t=1s
App (b): Total 25MB = (packetSize: 1000Bytes) × (#Packets: 25000)
with data-rate: 5Mbps app starts at t=15s
Simulation time: t = 0s ~ 120s
Two apps are running:
App (a): from Node 2 to Node 0 at port 8080
App (b): from Node 3 to Node 0 at port 8081

Scenario 3



Simulation time: t = 0s ~ 120s

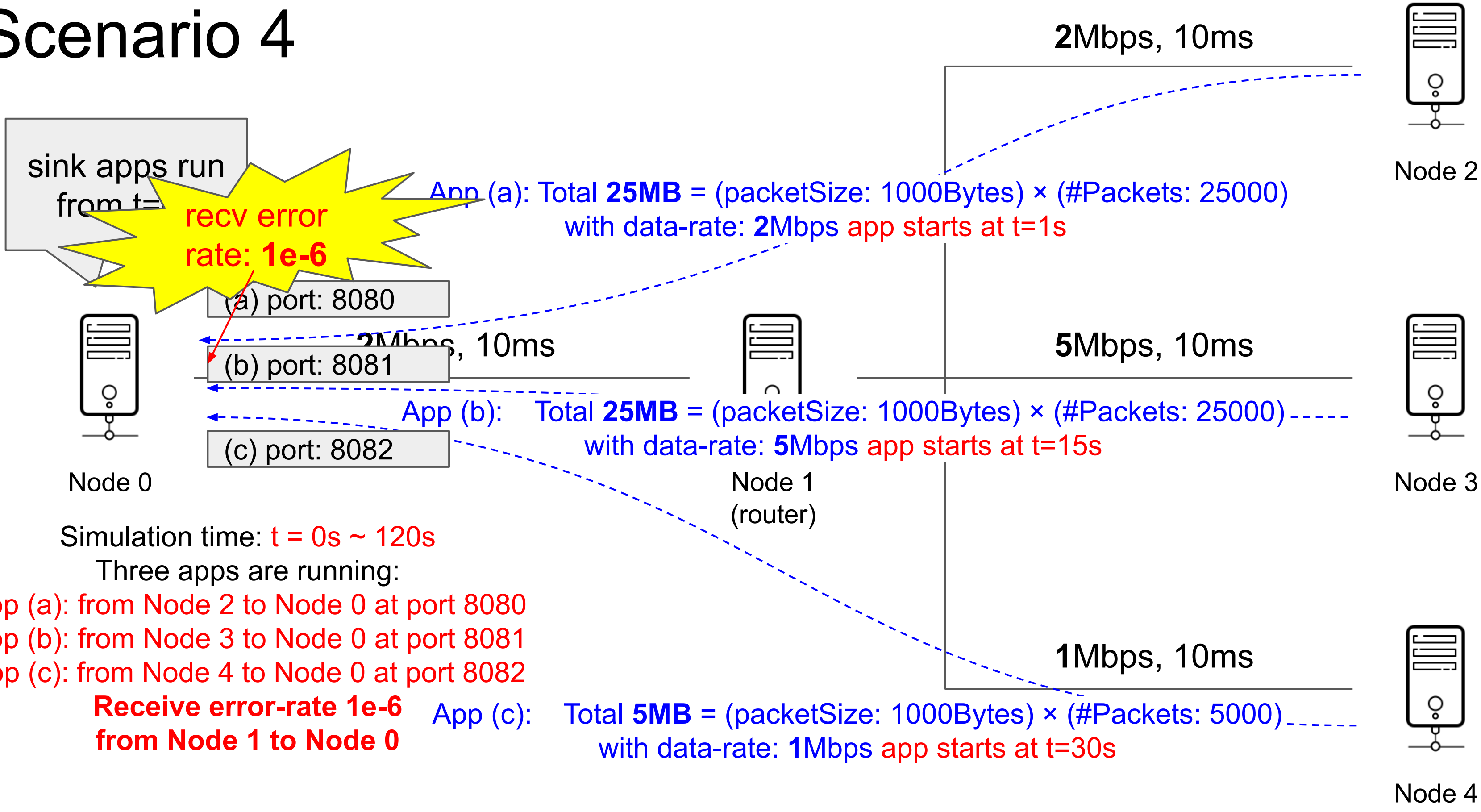
Three apps are running:

App (a): from Node 2 to Node 0 at port 8080

App (b): from Node 3 to Node 0 at port 8081

App (c): from Node 4 to Node 0 at port 8082

Scenario 4



Simulation time: t = 0s ~ 120s

Three apps are running:

App (a): from Node 2 to Node 0 at port 8080

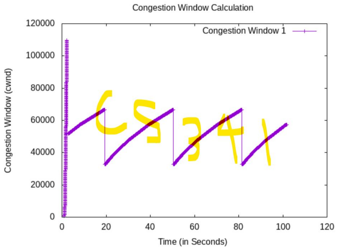
App (b): from Node 3 to Node 0 at port 8081

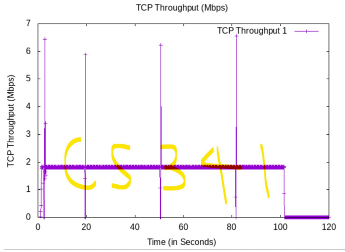
App (c): from Node 4 to Node 0 at port 8082

Receive error-rate 1e-6
from Node 1 to Node 0

Task 1

sample answers
from 1.1 →





1. Write a program code to simulate network system given in the previous slides.
We recommend you to use *gnuplot* to plot the graph. Copy the **prac3_skeleton.cc** file to **scratch/** directory and run by **“./waf --run scratch/prac3_skeleton”** to execute the code. **(30pts)**

1.1. Scenario 1: When app (a) is running (skeleton code). (2pts)

1.1.1. Plot the *cwnd* (congestion window) of app (a) on a graph. (1pts)

1.1.2. Plot the TCP throughput of app (a) in each 0.1 second. (unit x:sec, y:Mbps). (1pts)

1.2. Scenario 2: When app (a, b) are running. (6pts)

1.2.1. Plot *cwnd* of app (a, b) respectively on a graph. (2pts)

1.2.2. Plot the calculated TCP throughput of each app (a, b) in each 0.1 second. (unit x:sec, y:Mbps). (2pts)

1.2.3. Describe how the graph is different from **Scenario 1 (1.1.)** and discuss the reason why. (2pts)

Task 1

1.3. Scenario 3: When app (a, b, c) are running. (6pts)

1.3.1. Plot *cwnd* of app (a, b, c) respectively on a graph. (2pts)

1.3.2. Plot the calculated TCP throughput of each app (a, b, c) in each 0.1 second. (unit x:sec, y:Mbps). (2pts)

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

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. (6pts)
(※ Tip: use RateErrorModel, like in examples/tutorial/sixth.cc)

1.4.1. Plot *cwnd* of app (a, b, c) respectively on a graph. (2pts)

1.4.2. Plot the calculated TCP throughput of each app (a, b, c) in each 0.1 second. (unit x:sec, y:Mbps). (2pts)

1.4.3. Describe how the graph is different from **Scenario 3 (1.3.)** and discuss the reason why. (2pts)

1.5. Submit your final simulation code of **Scenario 4** with the filename of ***prac3_20xxxxxx_YourName.cc* (10pts)**

Task 2

2. Change TCP congestion control algorithm (20pts).

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**. (10pts)

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. (5pts)

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. (5pts)

Task 2

2.2. Roll back the previous changes in **2.1.** to the original. Try **Scenario 4 (1.4)** using another TCP congestion control algorithms in NS3. (※ Tip: In NS-3 document, you can see how to change default NS-3 TCP control algorithm: https://www.nsnam.org/docs/models/html/tcp.html.) (10pts)

2.2.1. Plot *cwnd* and *TCP throughput* graphs when the algorithm is set to *Veno*. (2pts)

2.2.2. Plot *cwnd* and *TCP throughput* graphs when the algorithm is set to *Yeah*. (2pts)

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. (6pts)

Total Received Bytes	NewReno	Veno	Yeah
app (a)			
app (b)			
app (c)			

Submit your report in **pdf format**, and your final simulation code of **Scenario 4** with the filename of ***prac3_20xxxxxx_YourName.cc.*** Submit a zip file on KLMS.
Due: 11/9 Mon, 11:59 pm. *Plagiarism & Late submission: 0 points.*

Appendix 1: tcp-congestion-ops.cc

```
void
TcpNewReno::CongestionAvoidance (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked)
{
  NS_LOG_FUNCTION (this << tcb << segmentsAcked);

  if (segmentsAcked > 0)
    {
      double adder = tcb->m_segmentSize;
      //double adder = static_cast<double> (tcb->m_segmentSize * tcb->m_segmentSize) / tcb->m_cWnd.Get ();
      adder = std::max (1.0, adder);
      tcb->m_cWnd += static_cast<uint32_t> (adder);
      NS_LOG_INFO ("In CongAvoid, updated to cwnd " << tcb->m_cWnd <<
                   " ssthresh " << tcb->m_ssThresh);
    }
}
```

Appendix 2: tcp-recovery-ops.cc

```
void
TcpClassicRecovery::EnterRecovery (Ptr<TcpSocketState> tcb, uint32_t dupAckCount,
                                   uint32_t unAckDataCount, uint32_t deliveredBytes)
{
  NS_LOG_FUNCTION (this << tcb << dupAckCount << unAckDataCount);
  NS_UNUSED (unAckDataCount);
  NS_UNUSED (deliveredBytes);
  tcb->m_cWnd = 1;
  tcb->m_cWndInfl = 1;
  //tcb->m_cWnd = tcb->m_ssThresh;
  //tcb->m_cWndInfl = tcb->m_ssThresh + (dupAckCount * tcb->m_segmentSize);
}
```

```
void
TcpClassicRecovery::DoRecovery (Ptr<TcpSocketState> tcb, uint32_t deliveredBytes)
{
  NS_LOG_FUNCTION (this << tcb << deliveredBytes);
  NS_UNUSED (deliveredBytes);
  //tcb->m_cWndInfl += tcb->m_segmentSize;
}

void
TcpClassicRecovery::ExitRecovery (Ptr<TcpSocketState> tcb)
{
  NS_LOG_FUNCTION (this << tcb);
  // Follow NewReno procedures to exit FR if SACK is disabled
  // (RFC2582 sec.3 bullet #5 paragraph 2, option 2)
  // For SACK connections, we maintain the cwnd = ssthresh. In fact,
  // this ACK was received in RECOVERY phase, not in OPEN. So we
  // are not allowed to increase the window
  //tcb->m_cWndInfl = tcb->m_ssThresh.Get ();
}
```

Useful material

- examples/tutorial/sixth.cc, seventh.cc https://youtu.be/9rkN3FtOkaQ
- examples/tcp/*.cc
- examples/routing/simple-global-routing.cc