Session 10

Parameterised point-to-point link and TCP throughput in Wireshark

The last session builds a link whose bandwidth, delay, loss rate, queue size and simulation time are command-line parameters, then measures TCP throughput from the pcap in Wireshark and compares it with the theoretical bound.

Objectives

× Do not copy. Read for understanding and the viva

- Complete questions 25 to 25 of the manual: parameterised point-to-point link and tcp throughput in wireshark
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

× Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q25	Create a point to pint network between two nodes with the following parameters	Complete

Preparation

× Do not copy. Read for understanding and the viva

- Use `CommandLine cmd; cmd.AddValue("bandwidth", ...)` for each parameter with the defaults given in the question.
- Loss other than queue drops comes from `RateErrorModel` attached to the receiving NetDevice with the given error rate.
- Queue size: `pointToPoint.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("10p"))`.
- In Wireshark: Statistics, Conversations, TCP, read bytes and duration; compute throughput with the formula below and compare with the bandwidth-delay product bound.

Question 25

Problem Statement

■ Write in lab record

Create a point to pint network between two nodes with the following parameters.

- Link bandwidth between the two nodes. Default is 5 Mbps.
- One way delay of the link. Default is 5 milliseconds.
- Loss rate of packets on the link. Default is 0.000001. (This covers losses other than those that occur due to buffer drops at node0.)
- Queue size of the buffer at node 0. Default is 10 packets.
- Simulation time. Default is 10 seconds.

Calculate the average TCP throughput at the receiver using Wireshark application for packet capturing.

Solution

Write in lab record

Steps

1. Copy the listing into `scratch/p2p-params.cc`. The five parameters are variables in `main` with the manual's defaults, registered with `CommandLine::AddValue`, so `./ns3 run "scratch/p2p-params --bandwidth=2Mbps --delay=20ms --lossRate=0.001 --queueSize=5 --simTime=20"` runs any variant.
2. Link: `PointToPointHelper` with `DataRate` set to `bandwidth` and `Delay` set to `delay`. Queue: `p2p.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("10p"))`, built from `queueSize`; the helper appends the `Packet` template type itself.
3. Loss: a `RateErrorModel` with `ErrorRate` equal to `lossRate` and `ErrorUnit` `ERROR_UNIT_PACKET`, attached as the `ReceiveErrorModel` of node 1's device. This drops each arriving packet independently with that probability, separate from the queue drops at node 0.
4. Traffic: `PacketSink` on node 1 port 8080 and `BulkSendHelper` on node 0 with `MaxBytes` 0, both running for the full `simTime`. Segment size 1448 bytes so each frame is exactly 1500 bytes of IP plus 2 bytes PPP; `TcpNewReno` so the formula sheet applies.
5. `p2p.EnablePcapAll("p2p-params")` writes `p2p-params-0-0.pcap` (sender) and `p2p-params-1-0.pcap` (receiver). The receiver file is the one the question asks about.
6. Run `./ns3 run scratch/p2p-params`, note the goodput it prints, then open the receiver pcap in Wireshark and follow the procedure below.

Configuration

How each parameter in the question maps to the script:

Parameter in the question	Option	Default	Where it is applied
Link bandwidth	`--bandwidth`	5Mbps	`PointToPointHelper::SetDeviceAttribute("DataRate", ...)` on both devices
One way delay	`--delay`	5ms	`PointToPointHelper::SetChannelAttribute("Delay", ...)`
Loss rate other than buffer drops	`--lossRate`	0.000001	`RateErrorModel` with `ErrorUnit` packet, `ReceiveErrorModel` of node 1's device
Queue size at node 0	`--queueSize`	10	`SetQueue("ns3::DropTailQueue", "MaxSize", "10p")`
Simulation time	`--simTime`	10	`Simulator::Stop` and the application stop times

`./ns3 run "scratch/p2p-params --PrintHelp"` lists these five options with their defaults, which is a quick way to show the examiner that the script is parameterised.

Program

p2p-params.cc

```cpp
/*
 * p2p-params.cc  --  MCSL-223 Session 10, question 25
 *
 * Purpose of the program:
 *   A two-node point-to-point link whose bandwidth, one-way delay, random
 *   packet loss rate, queue size at node 0 and simulation time are all
 *   command-line parameters with the defaults given in the manual.  A TCP
 *   bulk transfer runs from node 0 to node 1 and both interfaces are
 *   captured to pcap so the average TCP throughput can be read in
 *   Wireshark (Statistics, Conversations, TCP).
 *
 * Topology:
 *   n0 (BulkSend) ---- bandwidth, delay, DropTail queue, RateErrorModel ---- n1 (PacketSink :8080)
 *
 * Build and run (ns-3.36 or later):
 *   cp p2p-params.cc scratch/
 *   ./ns3 run scratch/p2p-params
 *   ./ns3 run "scratch/p2p-params --bandwidth=2Mbps --delay=20ms --lossRate=0.001 --queueSize=5 --simTime=2
 *   wireshark p2p-params-1-0.pcap
 */

#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("P2pParams");

/*
 * main: reads the five parameters, builds the link, attaches the loss
 * model and the queue, runs a TCP bulk transfer and prints the bytes the
 * sink received together with the throughput for comparison with Wireshark.
 */
int
main(int argc, char* argv[])
{
    std::string bandwidth = "5Mbps";  // link bandwidth
    std::string delay = "5ms";        // one-way delay
    double lossRate = 0.000001;       // random per-packet loss, not queue drops
    uint32_t queueSize = 10;          // packets in the DropTail queue at node 0
    double simTime = 10.0;            // seconds

    CommandLine cmd(__FILE__);
    cmd.AddValue("bandwidth", "Link bandwidth between the two nodes", bandwidth);
    cmd.AddValue("delay", "One way delay of the link", delay);
    cmd.AddValue("lossRate", "Loss rate of packets on the link (per packet)", lossRate);
    cmd.AddValue("queueSize", "Queue size of the buffer at node 0 in packets", queueSize);
    cmd.AddValue("simTime", "Simulation time in seconds", simTime);
    cmd.Parse(argc, argv);

    // Ethernet-sized segments; ns-3 defaults to 536 bytes
    Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(1448));
    // NewReno halves cwnd on loss, as in the formula sheet (ns-3.35+ defaults to Cubic)
    Config::SetDefault("ns3::TcpL4Protocol::SocketType", TypeIdValue(TcpNewReno::GetTypeId()));

    NodeContainer nodes;
    nodes.Create(2);
```

```
    PointToPointHelper p2p;
    p2p.SetDeviceAttribute("DataRate", StringValue(bandwidth));
    p2p.SetChannelAttribute("Delay", StringValue(delay));
    p2p.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue(std::to_string(queueSize) + "p"));
    NetDeviceContainer devices = p2p.Install(nodes);

    // random losses on the link, applied where node 1 receives
    Ptr<RateErrorModel> em = CreateObject<RateErrorModel>();
    em->SetAttribute("ErrorRate", DoubleValue(lossRate));
    em->SetAttribute("ErrorUnit", StringValue("ERROR_UNIT_PACKET"));
    devices.Get(1)->SetAttribute("ReceiveErrorModel", PointerValue(em));

    InternetStackHelper stack;
    stack.Install(nodes);

    Ipv4AddressHelper address;
    address.SetBase("10.1.1.0", "255.255.255.0");
    Ipv4InterfaceContainer ifaces = address.Assign(devices);

    uint16_t port = 8080;
    PacketSinkHelper sinkHelper("ns3::TcpSocketFactory",
                                InetSocketAddress(Ipv4Address::GetAny(), port));
    ApplicationContainer sinkApp = sinkHelper.Install(nodes.Get(1));
    sinkApp.Start(Seconds(0.0));
    sinkApp.Stop(Seconds(simTime));

    BulkSendHelper bulk("ns3::TcpSocketFactory", InetSocketAddress(ifaces.GetAddress(1), port));
    bulk.SetAttribute("MaxBytes", UintegerValue(0));  // keep sending until stopped
    ApplicationContainer srcApp = bulk.Install(nodes.Get(0));
    srcApp.Start(Seconds(0.0));
    srcApp.Stop(Seconds(simTime));

    // p2p-params-0-0.pcap (sender side) and p2p-params-1-0.pcap (receiver side)
    p2p.EnablePcapAll("p2p-params");

    Simulator::Stop(Seconds(simTime));
    Simulator::Run();


    Ptr<PacketSink> sink = DynamicCast<PacketSink>(sinkApp.Get(0));

    std::cout << "bandwidth=" << bandwidth << " delay=" << delay << " lossRate=" << lossRate

              << " queueSize=" << queueSize << "p simTime=" << simTime << "s" << std::endl;

    std::cout << "Sink received " << sink->GetTotalRx() << " bytes in " << simTime

              << " s: goodput " << sink->GetTotalRx() * 8.0 / simTime / 1e6 << " Mbit/s"

              << std::endl;



    Simulator::Destroy();

    return 0;

}
```

Diagram

```
n0 10.1.1.1                                                          n1 10.1.1.2
BulkSend :8080 ⟶ [DropTail 10 p] ==== 5 Mbps, 5 ms one way ==== [RateErrorModel 1e-6] ⟶ PacketSink
pcap: p2p-params-0-0.pcap                                            pcap: p2p-params-1-0.pcap
```

Output

Expected console output for the defaults (ns-3 is not installed here; the listing was checked by reading against the ns-3.36 helper API):

```
$ ./ns3 run scratch/p2p-params
bandwidth=5Mbps delay=5ms lossRate=1e-06 queueSize=10p simTime=10s
Sink received 5980240 bytes in 10 s: goodput 4.78419 Mbit/s
```

Wireshark procedure on `p2p-params-1-0.pcap`:

1. File, Open, choose `p2p-params-1-0.pcap`. The frames are shown as PPP; data segments are 1502 bytes, ACKs 54 bytes.
2. Statistics, Conversations, click the TCP tab. One row appears. Tick “Limit to display filter” only if you applied a filter.
3. Read the columns Address A, Port A, Address B, Port B, Packets, Bytes, Packets A to B, Bytes A to B, Rel Start, Duration, Bits/s A to B.
4. Statistics, Capture File Properties gives the same duration and the average bits per second for the whole file as a cross-check.
5. Optional: Statistics, TCP Stream Graphs, Throughput draws the per-second throughput; a flat line near 5 Mbit/s after the first 100 ms is what you should see.
6. Analyze, Expert Information lists every retransmission; with the default loss rate each one is a queue drop at node 0, and the count should be small (a few dozen in 10 s).

Expected Conversations row (TCP tab):

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A to B	Bytes A to B	Rel Start	Duration	Bits/s A to B
10.1.1.1	49153	10.1.1.2	8080	8262	6.43 MB	4131	6.20 MB	0.000	9.9986	4.96 Mbit/s

The same table from the command line, useful on a lab machine without a desktop. `tshark` ships with Wireshark; `-q -z conv,tcp` prints the TCP conversations and nothing else (expected format):

```
$ tshark -r p2p-params-1-0.pcap -q -z conv,tcp
================================================================================
TCP Conversations
Filter:<No Filter>
                                               |       ←      | |       →      | |     Total     |    Rela
                                               | Frames  Bytes | | Frames  Bytes | | Frames  Bytes |      St
10.1.1.1:49153        ↔ 10.1.1.2:8080            4131  223074     4131 6202762     8262 6425836     0.000000
================================================================================
```

Expected non-default run, ten times the delay, everything else default:

```
$ ./ns3 run "scratch/p2p-params --delay=50ms"
bandwidth=5Mbps delay=50ms lossRate=1e-06 queueSize=10p simTime=10s
Sink received 5562320 bytes in 10 s: goodput 4.44986 Mbit/s
```

Explanation

Average TCP throughput at the receiver from the formula sheet, using the Conversations figures (frame bytes A to B, so headers included):

$$\text{Throughput} = \frac{8 \times 6.20 \times 10^6}{9.9986} = 4.96 \times 10^6 \text{ bit/s}$$

The goodput that ns-3 prints counts payload only:

$$\text{Goodput} = \frac{8 \times 5{,}980{,}240}{10} = 4.78 \times 10^6 \text{ bit/s}$$

The two differ by the header share, 54 of every 1502 bytes, about 3.6 percent, plus the first few milliseconds of handshake and slow start.

Compare with the bound. The bandwidth-delay product of the link is

$$BDP = 5 \times 10^6 \times 0.010 = 50{,}000 \text{ bits} = 6250 \text{ bytes} \approx 4.2 \text{ segments}$$

TCP can fill the link only if its window is at least the BDP. Here the window is limited by the 10-packet queue at node 0: the largest window before a drop is the BDP plus the queue,

$$cwnd_{\max} \approx 6250 + 10 \times 1502 = 21{,}270 \text{ bytes}$$

and after a loss NewReno halves it to about 10.6 kB, still above the BDP. So the window never falls below the BDP, the link stays full, and the bound from the formula sheet is

$$\text{Throughput}_{\max} = \min\left(B, \frac{cwnd_{\max}}{RTT}\right) = \min\left(5 \text{ Mbit/s}, \frac{21{,}270 \times 8}{0.010}\right) = 5 \text{ Mbit/s}$$

The measured 4.96 Mbit/s is 99 percent of the bound; the missing 1 percent is the handshake and the first slow-start RTTs. The random loss rate hardly matters: with 4131 packets and a probability of one in a million, the expected number of random drops is 0.004, so in most runs there is none, and every drop you see in Wireshark (Analysis, Expert Information, or the filter `tcp.analysis.retransmission`) is a queue overflow at node 0.

If you rerun with `--delay=50ms` the RTT is 100 ms, the BDP is 62.5 kB, and the window (at most about 77 kB, averaging about 58 kB under the sawtooth) becomes the limit: the same Wireshark procedure then gives about 4.6 Mbit/s, and the record should show that the bound is now the window term rather than the link rate. Side by side:

Run	RTT	BDP	Largest window (BDP plus 10 frames)	Window after halving	Binding term	Expected Wireshark throughput
defaults, 5 ms	10 ms	6250 B	21.3 kB	10.6 kB, above BDP	link rate, 5 Mbit/s	4.96 Mbit/s
`--delay=50ms`	100 ms	62.5 kB	77.5 kB	38.8 kB, below BDP	window over RTT, about 4.6 Mbit/s	about 4.6 Mbit/s

The queue at node 0 is the same 10 packets in both runs; what changes is how large it is compared with the pipe.

Formula Sheet

× Do not copy. Read for understanding and the viva

Throughput

Bytes received at the sink divided by the time the flow was active, converted to bits per second:

$$\text{Throughput} = \frac{8 \times \text{bytes received}}{t_{\text{end}} - t_{\text{start}}} \text{ bit/s}$$

Bandwidth-delay product

The amount of data in flight on a link of capacity B and round-trip time RTT. A TCP window smaller than this cannot fill the link:

$$BDP = B \times RTT$$

For the Session 10 defaults, $B = 5\,\text{Mbit/s}$ and one-way delay $5\,\text{ms}$ give $RTT = 10\,\text{ms}$ and $BDP = 5 \times 10^6 \times 0.010 = 50{,}000\,\text{bits} \approx 6.25\,\text{kB}$.

Transmission and propagation delay

$$t_{\text{trans}} = \frac{L}{B}, \qquad t_{\text{prop}} = \frac{d}{v}, \qquad t_{\text{total}} = t_{\text{trans}} + t_{\text{prop}} + t_{\text{queue}} + t_{\text{proc}}$$

where L is packet size in bits, B link rate, d distance and v signal speed (about 2×10^8 m/s in copper or fibre).

Constant bit rate traffic

An OnOff application with packet size L bytes and rate R bit/s sends one packet every

$$\Delta t = \frac{8L}{R} \text{ seconds}$$

TCP congestion window

Slow start doubles the window every RTT until the threshold; congestion avoidance adds one segment per RTT; a loss halves it (TCP NewReno):

$$cwnd_{\text{ss}}(n) = 2^n\, MSS, \qquad cwnd_{\text{ca}} \leftarrow cwnd + \frac{MSS^2}{cwnd}, \qquad cwnd_{\text{loss}} \leftarrow \frac{cwnd}{2}$$

The maximum throughput of one TCP flow is bounded by

$$\text{Throughput}_{\text{max}} = \frac{cwnd_{\text{max}}}{RTT}$$

Packet loss

$$\text{Loss rate} = \frac{\text{packets sent} - \text{packets received}}{\text{packets sent}}$$

Viva Questions

× Do not copy. Read for understanding and the viva

- **Q:** How does the script take the five parameters? **A:** `CommandLine::AddValue(name, help, variable)` for each, then `cmd.Parse(argc, argv)`; `--PrintHelp` lists them with their defaults.
- **Q:** What does the 0.000001 loss rate model that the queue does not? **A:** Random corruption on the link. `RateErrorModel` drops each received packet with that probability, independent of the buffer at node 0.
- **Q:** Where is the 10-packet queue and what happens when it is full? **A:** In node 0's `PointToPointNetDevice`; the eleventh packet to arrive while ten are waiting is tail-dropped.
- **Q:** Which pcap do you open and why? **A:** `p2p-params-1-0.pcap`, node 1's interface, because the question asks for throughput at the receiver.
- **Q:** What is the difference between Bits/s A to B in Conversations and the goodput ns-3 prints? **A:** Wireshark counts frame bytes including PPP, IP and TCP headers; the sink counts payload only.
- **Q:** What is the bandwidth-delay product here and what does it decide? **A:** 5 Mbit/s times 10 ms, 6250 bytes. If the window stays above it the link is the bound; if the window falls below it the window is the bound.
- **Q:** Why 1448-byte segments? **A:** 1448 plus 32 bytes of TCP header with options plus 20 bytes IP is 1500, the usual MTU, which makes the frame arithmetic clean.

- **Q:** How would you confirm a drop was a queue drop and not a random loss? **A:** Set `--lossRate=0` and rerun; any retransmission that remains is a queue drop. Or trace the queue's `Drop` source.

Common Mistakes

× Do not copy. Read for understanding and the viva

- Passing `"10"` instead of `"10p"` to `MaxSize`; the queue size string needs the unit (p for packets, B for bytes).
- Attaching the `RateErrorModel` to node 0's device. The manual says the loss is on the link apart from node 0's buffer, so it belongs on the receiving device.
- Leaving `ErrorUnit` at its default of bytes, which makes 0.000001 a per-byte rate, about 1500 times more loss than intended.
- Reading Bytes (both directions) instead of Bytes A to B in Conversations, which adds the ACK stream to the throughput.
- Dividing by the simulation time instead of the Duration shown in Wireshark when the flow did not run the whole time.
- Comparing the measured throughput with the link rate only. The record must show the BDP and explain which of the two terms in the bound applies.

Session Summary

■ Write in lab record

- Source listing of `p2p-params.cc` with the header comment, the parameter table and the two-node diagram showing where the queue and the error model sit
- Console output for the defaults and for one non-default run
- The Wireshark procedure (File, Open; Statistics, Conversations, TCP) and the Conversations row copied into a table
- The throughput calculation from bytes and duration, and the goodput cross-check from the ns-3 output
- The bandwidth-delay product, the maximum window from the 10-packet queue, and the bound comparison
- One sentence on which drops were random and which were queue drops

Edit this page

Previous
Session 9

Next
Section 2 · Data Mining Lab (WEKA)