

Session 1

Point-to-point topology and UDP echo

The first session builds the smallest possible network in NS-3, two nodes on a point-to-point link, and runs a UDP client and server over it. Every later topology is this pattern repeated.

Objectives

✕ Do not copy. Read for understanding and the viva

- Complete questions 1 to 2 of the manual: point-to-point topology and udp echo
- 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
Q1	Create a simple point to point network topology using two nodes	Complete
Q2	Create a UdpClient and UdpServer nodes and communicate at a fixed data rate	Complete

Preparation

✕ Do not copy. Read for understanding and the viva

- Read `first.cc` in the NS-3 tutorial; it is the answer to question 1 with different parameters.
- Know the helper sequence: NodeContainer, PointToPointHelper (DataRate, Delay), InternetStackHelper, Ipv4AddressHelper, UdpEchoServerHelper and UdpEchoClientHelper.
- Fixed data rate for the client means setting MaxPackets, Interval and PacketSize attributes; compute the rate with the CBR formula below.

Question 1

Problem Statement

■ Write in lab record

Create a simple point to point network topology using two nodes.

Solution

■ Write in lab record

Steps

1. Install ns-3 (3.36 or later) and build it once: `./ns3 configure --enable-examples && ./ns3 build`.
2. Save the program below as `scratch/p2p_two_nodes.cc`. Anything in `scratch/` is compiled automatically.
3. Run `./ns3 run scratch/p2p_two_nodes`. The first run compiles the file; later runs start at once.
4. Draw the topology in the record: two boxes, one line, the link parameters and the two addresses.
5. List the pcap files with `ls session1-*.pcap`. They are empty here (no traffic) and fill up in question 2.

Program

The five helper calls are the skeleton of every NS-3 script: nodes, channel plus devices, protocol stack, addresses, then run.

p2p_two_nodes.cc

```
1  /*
2   * MCSL-223 Section 1, Session 1, Question 1
3   * Point-to-point topology with two nodes.
4   *
5   * Build: copy to ns-3.36+/scratch/ and run
6   *   ./ns3 run scratch/p2p_two_nodes
7   *
8   * Topology:
9   *   n0 ----- n1
10  *       5 Mbps, 2 ms
11  *   10.1.1.1       10.1.1.2
12  */
13  #include "ns3/core-module.h"
14  #include "ns3/network-module.h"
15  #include "ns3/internet-module.h"
16  #include "ns3/point-to-point-module.h"
17
18  using namespace ns3;
19
20  NS_LOG_COMPONENT_DEFINE("P2PTwoNodes");
21
22  /*
23   * main: creates two nodes, joins them with one point-to-point link,
24   * installs the IP stack, assigns 10.1.1.0/24 and prints what was built.
25   * No traffic is sent; question 2 adds the applications.
26   */
27  int
28  main(int argc, char* argv[])
29  {
30      std::string dataRate = "5Mbps";
31      std::string delay = "2ms";
32
33      CommandLine cmd(__FILE__);
34      cmd.AddValue("dataRate", "Link data rate", dataRate);
35      cmd.AddValue("delay", "Link propagation delay", delay);
36      cmd.Parse(argc, argv);
37
38      // 1. Nodes: the two end hosts.
39      NodeContainer nodes;
40      nodes.Create(2);
41
42      // 2. Channel and net devices: one full-duplex point-to-point link.
43      PointToPointHelper p2p;
```

```

44     p2p.SetDeviceAttribute("DataRate", StringValue(dataRate));
45     p2p.SetChannelAttribute("Delay", StringValue(delay));
46     NetDeviceContainer devices = p2p.Install(nodes);
47
48     // 3. Protocol stack: IPv4, UDP, TCP on both nodes.
49     InternetStackHelper stack;
50     stack.Install(nodes);
51
52     // 4. Addresses: one /24 subnet for the link.
53     Ipv4AddressHelper address;
54     address.SetBase("10.1.1.0", "255.255.255.0");
55     Ipv4InterfaceContainer interfaces = address.Assign(devices);
56
57     // Report the topology so the lab record has something to show.
58     std::cout << "Nodes created      : " << nodes.GetN() << std::endl;
59     std::cout << "Link              : " << dataRate << ", delay " << delay
60     for (uint32_t i = 0; i < nodes.GetN(); ++i)
61     {
62         std::cout << "n" << i << " address          : " << interfaces.GetAddre
63     }
64
65     // Write a pcap per device (session1-0-0.pcap, session1-1-0.pcap).
66     p2p.EnablePcapAll("session1");
67
68     Simulator::Stop(Seconds(10.0));
69     Simulator::Run();
70     std::cout << "Simulation finished at " << Simulator::Now().GetSeconds()
71     Simulator::Destroy();
72     return 0;
73 }

```

Output

Topology drawn for the record:

```

1           5 Mbit/s, 2 ms
2   n0 ----- n1
3   10.1.1.1           10.1.1.2
4   (10.1.1.0/24, one point-to-point channel)

```

Expected console output (NS-3 is not installed on the machine that wrote this page, so values come from the parameters, not from a run):

```
1 Nodes created      : 2
2 Link               : 5Mbps, delay 2ms
3 n0 address         : 10.1.1.1
4 n1 address         : 10.1.1.2
5 Simulation finished at 10 s
```

Change the link from the command line without editing the file: `./ns3 run "scratch/p2p_two_nodes --dataRate=10Mbps --delay=5ms"`.

Explanation

What each helper call creates, for the viva:

Call	Objects created	Key attributes
<code>NodeContainer::Create(2)</code>	two <code>Node</code> objects, ids 0 and 1	none
<code>PointToPointHelper::Install</code>	two <code>PointToPointNetDevice</code> , one <code>PointToPointChannel</code> , two <code>DropTailQueue</code>	<code>DataRate</code> (device), <code>Delay</code> (channel), <code>MaxSize</code> (queue, 100 packets)
<code>InternetStackHelper::Install</code>	<code>Ipv4L3Protocol</code> , <code>ArpL3Protocol</code> , <code>UdpL4Protocol</code> , <code>TcpL4Protocol</code> , loopback device, list routing	<code>IpForward</code> true
<code>Ipv4AddressHelper::Assign</code>	one <code>Ipv4Interface</code> per device with an address and mask	base 10.1.1.0, mask /24
<code>EnablePcapAll</code>	one pcap file per device	prefix <code>session1</code>

`NodeContainer::Create(2)` makes two empty nodes. `PointToPointHelper::Install` creates one `PointToPointNetDevice` on each node and a `PointToPointChannel` between them; `DataRate` lives on the device (it decides how long a packet takes to serialise) and `Delay` on the channel (propagation time). `InternetStackHelper` adds IPv4, ARP, UDP and TCP to both nodes. `Ipv4AddressHelper::Assign` hands out 10.1.1.1 and 10.1.1.2 in device order. Nothing is scheduled,

so `Simulator::Run` returns as soon as the stop event at 10 s fires. From the formula sheet, one 1024-byte packet on this link takes $t_{\text{trans}} = 8 \times 1054/5 \times 10^6 = 1.686 \text{ ms}$ to serialise (1024 bytes payload plus 8 UDP, 20 IP and 2 PPP header bytes) and $t_{\text{prop}} = 2 \text{ ms}$ to propagate, so $t_{\text{total}} = 3.686 \text{ ms}$; question 2 shows exactly that number in the log.

Question 2

Problem Statement

■ Write in lab record

Create a `UdpClient` and `UdpServer` nodes and communicate at a fixed data rate.

Solution

■ Write in lab record

Steps

1. Pick the fixed rate: packet size $L = 1024$ bytes and rate $R = 1 \text{ Mbit/s}$. From the formula sheet the client interval is $\Delta t = 8L/R = 8192/10^6 = 8.192 \text{ ms}$.
2. Save the program as `scratch/udp_echo_fixed_rate.cc` and run `./ns3 run scratch/udp_echo_fixed_rate`.
3. Read the client and server log lines; NS-3 prints them because the script enables `LOG_LEVEL_INFO` on both applications.
4. Open `session1-echo-1-0.pcap` in Wireshark (n1's device) and confirm 10 UDP packets in each direction with the filter `udp.port == 9`.
5. Try another rate: `./ns3 run "scratch/udp_echo_fixed_rate --rate=20000000 --maxPackets=20"` halves the interval to 4.096 ms.

Intervals for other fixed rates with a 1024-byte packet, from the same formula:

Rate R	Interval $8L/R$	Command line
500 kbit/s	16.384 ms	<code>--rate=500000</code>
1 Mbit/s	8.192 ms	default
2 Mbit/s	4.096 ms	<code>--rate=2000000</code>
4 Mbit/s	2.048 ms	<code>--rate=4000000</code> (80 percent of the 5 Mbit/s link)

Program

udp_echo_fixed_rate.cc

```
1  /*
2   * MCSL-223 Section 1, Session 1, Question 2
3   * UDP echo client and server on the two-node link, sending at a fixed rate.
4   *
5   * Build: copy to ns-3.36+/scratch/ and run
6   *   ./ns3 run scratch/udp_echo_fixed_rate
7   *
8   * Fixed data rate: packet size L = 1024 bytes, rate R = 1 Mbit/s,
9   * so the client interval is  $8L/R = 8192 / 1e6 = 8.192$  ms.
10  *
11  *   n0 (client) ----- n1 (server, port 9)
12  *   10.1.1.1   5 Mbps, 2 ms  10.1.1.2
13  */
14  #include "ns3/core-module.h"
15  #include "ns3/network-module.h"
16  #include "ns3/internet-module.h"
17  #include "ns3/point-to-point-module.h"
18  #include "ns3/applications-module.h"
19
20  using namespace ns3;
21
22  NS_LOG_COMPONENT_DEFINE("UdpEchoFixedRate");
23
24  /*
25   * main: builds the Session 1 link, puts a UdpEchoServer on n1 and a
26   * UdpEchoClient on n0 that sends maxPackets packets of packetSize bytes
27   * every  $8*packetSize/rate$  seconds, then runs for 10 s.
28   */
29  int
30  main(int argc, char* argv[])
31  {
32      uint32_t packetSize = 1024;    // bytes
33      uint32_t maxPackets = 10;
34      double rateBps = 1e6;          // bit/s, the fixed data rate
35
36      CommandLine cmd(__FILE__);
37      cmd.AddValue("packetSize", "UDP payload in bytes", packetSize);
38      cmd.AddValue("maxPackets", "Packets the client sends", maxPackets);
39      cmd.AddValue("rate", "Client data rate in bit/s", rateBps);
40      cmd.Parse(argc, argv);
41
42      // Print the client and server log lines (the expected output).
43      LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
```

```
44     LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
45
46     NodeContainer nodes;
47     nodes.Create(2);
48
49     PointToPointHelper p2p;
50     p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
51     p2p.SetChannelAttribute("Delay", StringValue("2ms"));
52     NetDeviceContainer devices = p2p.Install(nodes);
53
54     InternetStackHelper stack;
55     stack.Install(nodes);
56
57     Ipv4AddressHelper address;
58     address.SetBase("10.1.1.0", "255.255.255.0");
59     Ipv4InterfaceContainer interfaces = address.Assign(devices);
60
61     // Server on n1, listening on UDP port 9.
62     UdpEchoServerHelper echoServer(9);
63     ApplicationContainer serverApps = echoServer.Install(nodes.Get(1));
64     serverApps.Start(Seconds(1.0));
65     serverApps.Stop(Seconds(10.0));
66
67     // Client on n0. Interval = 8L/R seconds gives the fixed rate.
68     double intervalUs = 8.0 * packetSize / rateBps * 1e6;
69     UdpEchoClientHelper echoClient(interfaces.GetAddress(1), 9);
70     echoClient.SetAttribute("MaxPackets", UIntegerValue(maxPackets));
71     echoClient.SetAttribute("Interval", TimeValue(MicroSeconds(static_cast<u
72     echoClient.SetAttribute("PacketSize", UIntegerValue(packetSize));
73     ApplicationContainer clientApps = echoClient.Install(nodes.Get(0));
74     clientApps.Start(Seconds(2.0));
75     clientApps.Stop(Seconds(10.0));
76
77     std::cout << "Client rate " << rateBps / 1e6 << " Mbit/s, packet " << pa
78               << " B, interval " << intervalUs / 1000.0 << " ms, packets " <
79               << std::endl;
80
81     p2p.EnablePcapAll("session1-echo");
82
83     Simulator::Run();
84     Simulator::Destroy();
85     return 0;
86 }
```

Output

Expected output (first three and last exchanges shown; the remaining lines follow the same 8.192 ms spacing). Time stamps are computed from the link parameters: one-way delay is 1.686 ms serialisation plus 2 ms propagation, 3.686 ms.

```
1 Client rate 1 Mbit/s, packet 1024 B, interval 8.192 ms, packets 10
2 At time +2s client sent 1024 bytes to 10.1.1.2 port 9
3 At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
4 At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
5 At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
6 At time +2.00819s client sent 1024 bytes to 10.1.1.2 port 9
7 At time +2.01188s server received 1024 bytes from 10.1.1.1 port 49153
8 At time +2.01188s server sent 1024 bytes to 10.1.1.1 port 49153
9 At time +2.01556s client received 1024 bytes from 10.1.1.2 port 9
10 At time +2.01638s client sent 1024 bytes to 10.1.1.2 port 9
11 At time +2.02007s server received 1024 bytes from 10.1.1.1 port 49153
12 At time +2.02007s server sent 1024 bytes to 10.1.1.1 port 49153
13 At time +2.02376s client received 1024 bytes from 10.1.1.2 port 9
14 ...
15 At time +2.07373s client sent 1024 bytes to 10.1.1.2 port 9
16 At time +2.07741s server received 1024 bytes from 10.1.1.1 port 49153
17 At time +2.07741s server sent 1024 bytes to 10.1.1.1 port 49153
18 At time +2.0811s client received 1024 bytes from 10.1.1.2 port 9
```

Calculation to show in the record:

Quantity	Value
Packet size L	1024 bytes = 8192 bits
Fixed rate R	1 Mbit/s
Interval $\Delta t = 8L/R$	8.192 ms
One-way delay per packet	1.686 ms + 2 ms = 3.686 ms
Round trip seen by the client	7.37 ms
Bytes received at server	$10 \times 1024 = 10240$
Server receive window	2.00369 s to 2.07741 s = 73.7 ms
Measured throughput $8 \times 10240 / 0.0737$	1.11 Mbit/s

The measured value is 10/9 of the nominal 1 Mbit/s because ten packets span only nine intervals; with 1000 packets the ratio drops to 1.001.

Explanation

`UdpEchoServerHelper(9)` binds a UDP socket on port 9 of `n1` and echoes every datagram back. `UdpEchoClientHelper` on `n0` has three attributes that set the rate: `PacketSize` (bytes per datagram), `Interval` (time between datagrams) and `MaxPackets` (how many). Holding the size fixed and choosing the interval as $8L/R$ gives a constant bit rate of exactly R ; that is the same idea `OnOffHelper` uses internally in later sessions. The server starts at 1 s and the client at 2 s so the socket is listening before the first datagram arrives. Every log line carries the simulator time, which is why the record can show the transmission and propagation delay from the formula sheet appearing in the output.

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** What is the difference between a Node, a NetDevice and a Channel? **A:** A Node is the computer, a NetDevice is its network card plus driver, a Channel is the wire between two devices.

- **Q:** Where is DataRate set and where is Delay set, and why? **A:** DataRate on the device because serialisation happens in the card; Delay on the channel because propagation happens on the wire.
- **Q:** Why does the server receive the first packet at 2.00369 s and not at 2.002 s? **A:** 2 ms is propagation only; 1054 bytes at 5 Mbit/s add 1.686 ms of serialisation.
- **Q:** How do you send at a fixed rate with UdpEchoClient? **A:** Fix PacketSize and set Interval to 8L/R seconds.
- **Q:** What does `Ipv4AddressHelper::Assign` return? **A:** An `Ipv4InterfaceContainer`; `GetAddress(i)` gives the address of the i-th device in the container.
- **Q:** Why start the server before the client? **A:** A datagram that arrives before the server socket is bound is dropped.
- **Q:** What is in `session1-echo-0-0.pcap`? **A:** Every frame that entered or left device 0 of node 0, in libpcap format readable by Wireshark.
- **Q:** Why does the client use port 49153? **A:** It is the first ephemeral port NS-3 allocates when a socket is bound without a port.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Passing 8.192 to `Milliseconds()`; it takes an integer and silently rounds to 8 ms. Use `MicroSeconds(8192)`.
- Starting the client before or at the same time as the server, so the first datagram is lost.
- Forgetting `LogComponentEnable`, then reporting that “nothing happens” because the run prints nothing.
- Reading the pcap of the wrong node; the file name is prefix-nodeld-deviceld.
- Editing files under `examples/` instead of `scratch/`; only `scratch/` picks up new files without touching the build files.
- Omitting the topology diagram from the record; the manual requires it with every program.

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 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

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

where L is packet size in bits, B link rate, d distance and v signal speed (about $2 \times 10^8 \text{ m/s}$ 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):

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

The maximum throughput of one TCP flow is bounded by

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

Packet loss

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

Session Summary

■ Write in lab record

- `p2p_two_nodes.cc` listing with the header comment, the topology diagram and the address printout
- `udp_echo_fixed_rate.cc` listing, the 40 echo log lines and the rate table (interval 8.192 ms for 1 Mbit/s, throughput 1.11 Mbit/s)
- Wireshark screenshot of `session1-echo-1-0.pcap` filtered on `udp.port == 9`

 Edit this page

 [Previous](#)
Section 1 · Computer Networks Lab (NS-3)

[Next](#) 
Session 2