

Session 9

Saturating the bottleneck and plotting cwnd

Raising the UDP rate until it fills the whole bridge starves TCP completely. The plot of congestion window against time, annotated with both UDP rates, is the deliverable.

Objectives

✕ Do not copy. Read for understanding and the viva

- Complete questions 23 to 24 of the manual: saturating the bottleneck and plotting cwnd
- 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
Q23	In the last session 8, Increase the UDP rate at 30 second to Rate2 such that it clogs...	Complete
Q24	Use MatplotlibLib or GNUPlot to visualize cwnd vs time, also mention Rate1 and Rate2	Complete

Preparation

✕ Do not copy. Read for understanding and the viva

- Schedule the rate change with `Simulator::Schedule(Seconds(30.0), ...)` that sets the OnOff application's DataRate attribute to Rate2.
- Plot with matplotlib: time on x, cwnd in bytes on y, vertical lines at 20 s and 30 s labelled Rate1 and Rate2.

- Explain each phase of the curve in the record: slow start, steady state, halving after 20 s, collapse after 30 s.

Question 23

Problem Statement

■ Write in lab record

In the last session 8, Increase the UDP rate at 30 second to Rate2 such that it clogs whole of the dumbbell bridge capacity.

Solution

■ Write in lab record

`dumbbell-rate2.cc` is the Session 8 script with one scheduled event added and the run extended to 45 s. Bridge 1 Mbit/s, Rate1 500 kbit/s from 20 s, Rate2 1 Mbit/s from 30 s.

Steps

1. Copy the listing into `scratch/dumbbell-rate2.cc` and run `./ns3 run scratch/dumbbell-rate2`.
2. The UDP `OnOffApplication` is installed once with `DataRate Rate1`. Its `DataRate` attribute can be changed while it runs; the application reads it every time it schedules the next packet.
3. `ChangeRate(Ptr<Application> app, DataRate rate)` calls `app->SetAttribute("DataRate", DataRateValue(rate))` and prints the time. It is scheduled with `Simulator::Schedule(Seconds(30.0), &ChangeRate, udpSrc.Get(0), DataRate("1Mbps"))`.
4. Rate1, Rate2 and both times are command-line options (`--rate1`, `--rate2`, `--udpStart`, `--rate2Time`), so the same binary can show the examiner other cases.
5. The cwnd trace is identical to Session 8: connect at 1.001 s to `/NodeList/0/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow` and write `time cwnd` to `cwnd.txt`.

Program

dumbbell-rate2.cc

```

1  /*
2  * dumbbell-rate2.cc  --  MCSL-223 Session 9, questions 23 and 24
3  *
4  * Purpose of the program:
5  *   The Session 8 dumbbell, extended: TCP starts at 1 s, UDP starts at 20 s
6  *   at Rate1 (500 kb/s, half the 1 Mbit/s bridge) and at 30 s the UDP rate
7  *   is raised to Rate2 (1 Mbit/s, the whole bridge) by a scheduled event
8  *   that changes the OnOff application's DataRate attribute (Q23). The
9  *   congestion window of the TCP socket is traced to cwnd.txt and plotted
10 *   with plot_cwnd.py, which marks Rate1 and Rate2 (Q24).
11 *
12 * Topology (Session 2 dumbbell, all links point-to-point):
13 *   n0 (TCP src) --10Mbps,1ms--\                               /--10Mbps,1ms-- n4 (TC
14 *                                   n2 --1Mbps,10ms-- n3
15 *   n1 (UDP src) --10Mbps,1ms--/   (the bridge)   \--10Mbps,1ms-- n5 (UDP
16 *
17 * Build and run (ns-3.36 or later):
18 *   cp dumbbell-rate2.cc scratch/
19 *   ./ns3 run scratch/dumbbell-rate2
20 *   python3 plot_cwnd.py cwnd.txt cwnd.png
21 */
22
23 #include "ns3/applications-module.h"
24 #include "ns3/core-module.h"
25 #include "ns3/internet-module.h"
26 #include "ns3/network-module.h"
27 #include "ns3/point-to-point-module.h"
28
29 using namespace ns3;
30
31 NS_LOG_COMPONENT_DEFINE("DumbbellRate2");
32
33 /*
34 * CwndChange: trace sink for the CongestionWindow attribute of the TCP
35 * socket. Writes "time newCwnd" (bytes) on every change.
36 */
37 static void
38 CwndChange(Ptr<OutputStreamWrapper> stream, uint32_t oldCwnd, uint32_t newCw
39 {
40     (void)oldCwnd;
41     *stream->GetStream() << Simulator::Now().GetSeconds() << " " << newCwnd
42 }
43

```

```

44  /*
45   * TraceCwnd: connects CwndChange to the first TCP socket of node 0.
46   * Scheduled at 1.001 s, just after BulkSend has created the socket.
47   */
48  static void
49  TraceCwnd(Ptr<OutputStreamWrapper> stream)
50  {
51      Config::ConnectWithoutContext("/NodeList/0/$ns3::TcpL4Protocol/SocketLis
52                                  MakeBoundCallback(&CwndChange, stream));
53  }
54
55  /*
56   * ChangeRate: scheduled at 30 s. Sets the DataRate attribute of the
57   * running OnOff application; the next packet interval uses the new rate.
58   */
59  static void
60  ChangeRate(Ptr<Application> app, DataRate rate)
61  {
62      app->SetAttribute("DataRate", DataRateValue(rate));
63      std::cout << Simulator::Now().GetSeconds() << " s: UDP rate changed to "
64                << std::endl;
65  }
66
67  /*
68   * main: builds the dumbbell, installs the flows, schedules the rate change
69   * and the cwnd tracer, and prints the totals at the end.
70   */
71  int
72  main(int argc, char* argv[])
73  {
74      std::string bridgeRate = "1Mbps";
75      std::string rate1 = "500kb/s"; // half of the bridge
76      std::string rate2 = "1Mbps";   // whole bridge
77      double udpStart = 20.0;
78      double rate2Time = 30.0;
79      double simTime = 45.0;
80
81      CommandLine cmd(__FILE__);
82      cmd.AddValue("bridgeRate", "Data rate of the n2-n3 bridge", bridgeRate);
83      cmd.AddValue("rate1", "UDP rate from udpStart (Rate1)", rate1);
84      cmd.AddValue("rate2", "UDP rate from rate2Time (Rate2)", rate2);
85      cmd.AddValue("udpStart", "Time in seconds at which UDP starts", udpStart);
86      cmd.AddValue("rate2Time", "Time in seconds at which UDP switches to Rate", rate2Time);
87      cmd.AddValue("simTime", "Simulation time in seconds", simTime);

```

```
88 cmd.Parse(argc, argv);
89
90 Config::SetDefault("ns3::TcpSocket::SegmentSize", UIntegerValue(1000));
91 // NewReno halves cwnd on loss, as in the formula sheet (ns-3.35+ default)
92 Config::SetDefault("ns3::TcpL4Protocol::SocketType", TypeIdValue(TcpNewR
93
94 // ---- nodes: left first so the TCP source is NodeList/0 -----
95 NodeContainer left;
96 left.Create(2); // n0, n1
97 NodeContainer routers;
98 routers.Create(2); // n2, n3
99 NodeContainer right;
100
101 right.Create(2); // n4, n5
102
103 1
104 10
105 2
106 PointToPointHelper access;
107
108 3
109 access.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
110
111 4
112 access.SetChannelAttribute("Delay", StringValue("1ms"));
113
114 5
115 10
116 6
117 PointToPointHelper bridge;
118
119 7
120 bridge.SetDeviceAttribute("DataRate", StringValue(bridgeRate));
121
122 8
123 bridge.SetChannelAttribute("Delay", StringValue("10ms"));
124
125 9
126 11
127 0
128 NetDeviceContainer d02 = access.Install(left.Get(0), routers.Get(0));
129
130 11
131 1
132 NetDeviceContainer d12 = access.Install(left.Get(1), routers.Get(0));
133
134 11
135 2
136 NetDeviceContainer d23 = bridge.Install(routers.Get(0), routers.Get(1));
137
138 11
139 3
140 NetDeviceContainer d34 = access.Install(routers.Get(1), right.Get(0));
141
142 11
143 4
144 NetDeviceContainer d35 = access.Install(routers.Get(1), right.Get(1));
145
146 11
147 5
```

```
11
6   InternetStackHelper stack;
11
7   stack.Install(left);
11
8   stack.Install(routers);
11
9   stack.Install(right);
12
0
12
1   Ipv4AddressHelper address;
12
2   address.SetBase("10.1.1.0", "255.255.255.0");
12
3   address.Assign(d02);
12
4   address.SetBase("10.1.2.0", "255.255.255.0");
12
5   address.Assign(d12);
12
6   address.SetBase("10.1.3.0", "255.255.255.0");
12
7   address.Assign(d23);
12
8   address.SetBase("10.1.4.0", "255.255.255.0");
12
9   Ipv4InterfaceContainer i34 = address.Assign(d34);
13
0   address.SetBase("10.1.5.0", "255.255.255.0");
13
1   Ipv4InterfaceContainer i35 = address.Assign(d35);
13
2
13
3   Ipv4GlobalRoutingHelper::PopulateRoutingTables();
13
4
13
5   // ---- TCP n0 → n4 from 1 s -----
13
6   uint16_t tcpPort = 8080;
13
7   PacketSinkHelper tcpSinkHelper("ns3::TcpSocketFactory",
```

```
13
8           InetSocketAddress(Ipv4Address::GetAny(),
13
9   ApplicationContainer tcpSink = tcpSinkHelper.Install(right.Get(0));
14
0   tcpSink.Start(Seconds(0.0));
14
1   tcpSink.Stop(Seconds(simTime));
14
2
14
3   BulkSendHelper bulk("ns3::TcpSocketFactory", InetSocketAddress(i34.GetAd
14
4   bulk.SetAttribute("MaxBytes", UintegerValue(0));
14
5   ApplicationContainer tcpSrc = bulk.Install(left.Get(0));
14
6   tcpSrc.Start(Seconds(1.0));
14
7   tcpSrc.Stop(Seconds(simTime));
14
8
14
9   // ---- UDP n1 → n5 at Rate1 from 20 s -----
15
0   uint16_t udpPort = 9000;
15
1   PacketSinkHelper udpSinkHelper("ns3::UdpSocketFactory",
15
2           InetSocketAddress(Ipv4Address::GetAny(),
15
3   ApplicationContainer udpSink = udpSinkHelper.Install(right.Get(1));
15
4   udpSink.Start(Seconds(0.0));
15
5   udpSink.Stop(Seconds(simTime));
15
6
15
7   OnOffHelper onoff("ns3::UdpSocketFactory", InetSocketAddress(i35.GetAddr
15
8   onoff.SetConstantRate(DataRate(rate1), 1000);
15
9   ApplicationContainer udpSrc = onoff.Install(left.Get(1));
```

```

16
0    udpSrc.Start(Seconds(udpStart));
16
1    udpSrc.Stop(Seconds(simTime));
16
2
16
3    // ---- Q23: switch the same OnOff application to Rate2 at 30 s -----
16
4    Simulator::Schedule(Seconds(rate2Time), &ChangeRate, udpSrc.Get(0), Data
16
5
16
6    // ---- Q24: cwnd trace, connected just after the socket exists -----
16
7    AsciiTraceHelper ascii;
16
8    Simulator::Schedule(Seconds(1.001), &TraceCwnd, ascii.CreateFileStream("
16
9
17
0    Simulator::Stop(Seconds(simTime));
17
1    Simulator::Run();
17
2
17
3    // ---- report -----
17
4    Ptr<PacketSink> ts = DynamicCast<PacketSink>(tcpSink.Get(0));
17
5    Ptr<PacketSink> us = DynamicCast<PacketSink>(udpSink.Get(0));
17
6    std::cout << "TCP sink n4: " << ts->GetTotalRx() << " bytes over " << si
17
7        << " s = " << ts->GetTotalRx() * 8.0 / (simTime - 1.0) / 1000.
17
8        << std::endl;
17
9    std::cout << "UDP sink n5: " << us->GetTotalRx() << " bytes over " << si
18
0        << " s = " << us->GetTotalRx() * 8.0 / (simTime - udpStart) /
18
1        << std::endl;

```

```

18
2
18
3     Simulator::Destroy();
18
4     return 0;
18
5 }

```

Configuration

The defaults reproduce the question. These runs are worth keeping in the record because the examiner can ask what happens if Rate2 is not the whole bridge:

Command	Rate2 on the wire	Expected cwnd after 30 s
<code>./ns3 run scratch/dumbbell-rate2</code>	1.03 Mbit/s, above the bridge	resets to 1 kB, stays there
<code>./ns3 run "scratch/dumbbell-rate2 --rate2=750kb/s"</code>	0.77 Mbit/s	sawtooth between about 10 kB and 20 kB, TCP keeps about 230 kbit/s
<code>./ns3 run "scratch/dumbbell-rate2 --rate2=500kb/s"</code>	same as Rate1	no change at 30 s; the plot shows only the 20 s step
<code>./ns3 run "scratch/dumbbell-rate2 --rate2Time=25 --simTime=40"</code>	1.03 Mbit/s from 25 s	collapse five seconds earlier; move the second marker in the plot script

Diagram

```

1    n0 (TCP BulkSend from 1 s) --10 Mbps, 1 ms--\
2                                     n2 ---- 1 Mbps, 10 ms bridge
3    n1 (UDP OnOff: Rate1 at 20 s,  --10 Mbps, 1 ms--/      DropTail 100 pac
4    Rate2 at 30 s)

```

Output

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

```
1 $ ./ns3 run scratch/dumbbell-rate2
2 30 s: UDP rate changed to 10000000bps
3 TCP sink n4: 2880000 bytes over 44 s = 523.636 kbit/s
4 UDP sink n5: 2455000 bytes over 25 s = 785.6 kbit/s
```

Expected `wnd.txt` around the change:

```
1 $ awk '$1>29.5 && $1<34' wnd.txt
2 29.6104 44000
3 29.6112 22000
4 30.4257 23000
5 30.9917 1000
6 31.9917 2000
7 32.4133 1000
8 34.4133 1000
```

Explanation

At Rate2 the UDP source offers 1000-byte datagrams every 8 ms. On the wire each is 1030 bytes, so the offered load is

$$\frac{1030 \times 8}{0.008} = 1.03 \text{ Mbit/s}$$

slightly more than the bridge can carry even with no TCP at all. The queue at n2 is now permanently full of UDP datagrams. A TCP segment is accepted only if it arrives in the short gap after a departure and before the next UDP arrival, so almost every TCP segment is tail-dropped, and even the UDP flow loses about 3 percent (979 of 1000 kbit/s delivered). TCP sees repeated losses without enough duplicate ACKs for fast retransmit, so it falls back to retransmission timeouts: cwnd is reset to one segment, the timeout doubles after each failure, and the flow delivers almost nothing. Between 30 s and 45 s the TCP sink gains only about 30 kB, against 2.27 MB in the first 19 s and 580 kB during the Rate1 phase. The UDP average printed over 25 s mixes the two phases: 10 s at about 495 kbit/s and 15 s at about 979 kbit/s.

Question 24

Problem Statement

■ Write in lab record

Use Matplotlib or GNUPlot to visualize cwnd vs time, also mention Rate1 and Rate2.

Solution

■ Write in lab record

Steps

1. Install matplotlib once: `pip install matplotlib`.
2. Run `python3 plot_cwnd.py cwnd.txt cwnd.png` in the ns-3 directory after the simulation. The script reads the two-column trace, draws the window as a step plot and adds two dashed vertical lines at 20 s and 30 s labelled Rate1 and Rate2.
3. For gnuplot instead: `gnuplot plot_cwnd.gp`. It uses `with steps`, two `set arrow ... nohead` lines at 20 and 30 and `set label` for the rates; the output file is the same `cwnd.png`.
4. Paste `cwnd.png` in the record with the phase table below written under it.

Program

■ Lab record: every tab is one file of the answer. Write all of them.

plot_cwnd.py

plot_cwnd.py

```
1 #!/usr/bin/env python3
2 """plot_cwnd.py -- MCSL-223 Session 9, Q24.
3
4 Reads cwnd.txt ("time cwnd" per line, written by dumbbell-rate2.cc) and
5 draws the congestion window against time with vertical markers at the two
6 UDP rate changes.
7
8 Usage:  python3 plot_cwnd.py [cwnd.txt] [cwnd.png]
9 Needs:  matplotlib (pip install matplotlib)
10 """
11
12 import sys
13
14 import matplotlib
15
16 matplotlib.use("Agg") # write a file, no display needed
17 import matplotlib.pyplot as plt # noqa: E402
18
19 RATE1_TIME, RATE1 = 20.0, "Rate1 = 500 kbit/s (half the bridge)"
20 RATE2_TIME, RATE2 = 30.0, "Rate2 = 1 Mbit/s (whole bridge)"
21
22
23 def read_trace(path):
24     """Return two lists, time in seconds and cwnd in bytes."""
25     times, cwnds = [], []
26     with open(path) as f:
27         for line in f:
28             parts = line.split()
29             if len(parts) == 2:
30                 times.append(float(parts[0]))
31                 cwnds.append(int(parts[1]))
32     return times, cwnds
33
34
35 def main():
36     src = sys.argv[1] if len(sys.argv) > 1 else "cwnd.txt"
37     out = sys.argv[2] if len(sys.argv) > 2 else "cwnd.png"
38     times, cwnds = read_trace(src)
39     if not times:
40         sys.exit(f"{src}: no samples found")
41
42     top = max(cwnds)
43     plt.figure(figsize=(9, 5))
```

```
44 plt.step(times, cwnds, where="post", lw=1.2, label="cwnd (bytes)")
45 plt.axvline(RATE1_TIME, color="tab:orange", ls="--")
46 plt.text(RATE1_TIME + 0.3, top * 0.95, RATE1, color="tab:orange")
47 plt.axvline(RATE2_TIME, color="tab:red", ls="--")
48 plt.text(RATE2_TIME + 0.3, top * 0.85, RATE2, color="tab:red")
49 plt.xlabel("Time (s)")
50 plt.ylabel("Congestion window (bytes)")
51 plt.title("TCP cwnd on the 1 Mbit/s dumbbell bridge under UDP load")
52 plt.grid(alpha=0.3)
53 plt.legend(loc="upper left")
54 plt.tight_layout()
55 plt.savefig(out, dpi=120)
56 print(f"wrote {out}: {len(times)} samples, max cwnd {top} bytes, last {c")
57
58
59 if __name__ == "__main__":
60     main()
```

Output

The script was run here on a synthetic `cwnd.txt` with the expected shape (matplotlib 3.x, Python 3.13) to check that it draws and labels correctly:

```
1 $ python3 plot_cwnd.py cwnd.txt cwnd.png
2 wrote cwnd.png: 981 samples, max cwnd 105000 bytes, last 3000 bytes
```

Expected figure: a step curve of cwnd in bytes against time from 1 s to 45 s, an orange dashed line at 20 s labelled "Rate1 = 500 kbit/s (half the bridge)" and a red dashed line at 30 s labelled "Rate2 = 1 Mbit/s (whole bridge)".

Explanation

Phase by phase, tie each part of the curve to the formula sheet:

Phase	Time	What the curve does	Formula-sheet rule
Slow start	1.0 to 1.9 s	10 kB doubling each RTT to about 110 kB	cwnd doubles every RTT
First loss	about 1.9 s	drop to about 55 kB	queue of 100 packets overflows; cwnd halves
Congestion avoidance	2 to 20 s	slow straight climb to about 82 kB, no loss	plus one segment per RTT; RTT is about 0.5 s because of the queue
Rate1	20 to 30 s	halves at about 20.7 s, then a sawtooth between about 25 kB and 45 kB	UDP takes half the bridge and half the queue; TCP loses every few seconds and halves each time
Rate2	30 to 45 s	halves once more, then collapses to 1 kB with rare steps to 2 kB	queue is always full of UDP; every loss is a timeout, cwnd resets to one segment and the timeout doubles

Two points to make in the viva. First, during phases 2 and 3 the TCP throughput is constant even though cwnd changes a lot, because cwnd is above the 3000-byte bandwidth-delay product; the window only decides how deep the queue is. Second, the throughput bound in the formula sheet, $cwnd \text{ over RTT}$, explains the collapse: with cwnd at 1000 bytes and an RTT that has grown to the retransmission timeout (a second or more), the bound is under 8 kbit/s, which is what the sink sees.

Why UDP wins: the OnOff source has no feedback loop. TCP measures loss and reduces its window; UDP measures nothing and keeps its rate, so at Rate2 it takes the bridge and TCP gets only the gaps. This is the argument for congestion control, fair queueing at routers, or rate limits on UDP applications.

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

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

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** How is the UDP rate changed at 30 s without a second application? **A:** A scheduled event calls `SetAttribute("DataRate", ...)` on the running `OnOffApplication`; it reads the attribute when it schedules each next packet.
- **Q:** What are Rate1 and Rate2 and where do they come from? **A:** Half and all of the bridge capacity: 500 kbit/s and 1 Mbit/s for a 1 Mbit/s bridge.
- **Q:** Why does UDP at exactly 1 Mbit/s still overload a 1 Mbit/s link? **A:** The rate counts payload; UDP, IP and PPP headers add 30 bytes per 1000-byte datagram, so the wire load is 1.03 Mbit/s.
- **Q:** Why does cwnd go to 1000 bytes after 30 s instead of halving? **A:** With nearly every segment dropped there are no duplicate ACKs, so the loss is detected by timeout, which resets cwnd to one segment.
- **Q:** What does `plt.step` show that `plt.plot` would not? **A:** cwnd changes in jumps at ACK or loss events; a step plot draws it as it is, a line plot draws false slopes.
- **Q:** Why is the TCP throughput flat between 2 s and 20 s while cwnd climbs? **A:** cwnd is above the bandwidth-delay product, so the bridge is already full; extra window only fills the queue.
- **Q:** How would you make TCP survive Rate2? **A:** A queue discipline that isolates flows (fair queueing, `FqCoDel` in ns-3) or a smaller shared queue with early drops (RED, CoDel) so UDP cannot monopolise the buffer.
- **Q:** Where is the bandwidth-delay product of the bridge? **A:** 1 Mbit/s times 24 ms, 3000 bytes, three segments of 1000 bytes.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Installing a second OnOff application at 30 s instead of changing the rate of the first; the two sources then overlap and the offered load is Rate1 plus Rate2.

- Passing the attribute as `StringValue("1Mbps")` to a function that expects `DataRate`; both work with `SetAttribute`, but mixing them in `Simulator::Schedule` arguments gives a compile error about the callback signature.
- Running only to 40 s, which leaves too little of the Rate2 phase to show the collapse; use 45 s or more.
- Drawing the marker lines but not labelling them; the question explicitly asks to mention Rate1 and Rate2 on the plot.
- Describing the Rate2 phase as "cwnd halves" when it actually resets to one segment on timeout; the record should say which loss detection happened.
- Plotting cwnd in segments after tracing it in bytes without saying so; the axis label must match the trace.

Session Summary

■ Write in lab record

- Source listing of `dumbbell-rate2.cc` with the header comment and the dumbbell diagram marking Rate1 at 20 s and Rate2 at 30 s
- Run output with the rate-change line and the TCP and UDP totals
- `cwnd.txt` excerpt around 30 s showing the reset to 1000 bytes
- `plot_cwnd.py` (or `plot_cwnd.gp`) and the figure `cwnd.png` with both vertical markers labelled
- The five-phase table linking each part of the curve to the formula sheet
- The wire-load calculation showing why Rate2 exceeds the bridge

[✎ Edit this page](#)

[< Previous](#)
Session 8

Session 10 [Next >](#)