

Session 6

Fixed-distance ad-hoc nodes and CBR traffic

Placing nodes at exact 3D coordinates lets you study how distance affects a wireless link. A constant bit rate source then gives a steady, predictable load to measure.

Objectives

✕ Do not copy. Read for understanding and the viva

- Complete questions 13 to 15 of the manual: fixed-distance ad-hoc nodes and cbr traffic
- 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
Q13	Use 2 nodes to setup a wireless ad-hoc network where nodes are placed at a fixed...	Complete
Q14	Install UDP server and Client at these two nodes	Complete
Q15	Setup a CBR transmission between these nodes	Complete

Preparation

✕ Do not copy. Read for understanding and the viva

- Use `ConstantPositionMobilityModel` with a `ListPositionAllocator` and `Vector(x, y, z)` for each node.

- CBR is an OnOffHelper with `OnTime` constant 1 and `OffTime` constant 0, plus `DataRate` and `PacketSize`.
- Repeat at two or three distances and record delivered bytes; relate the drop to the propagation loss model.

Question 13

Problem Statement

■ Write in lab record

Use 2 nodes to setup a wireless ad-hoc network where nodes are placed at a fixed distance in a 3D plane.

Solution

■ Write in lab record

One script, `adhoc-cbr.cc`, answers questions 13 to 15. Question 13 is the topology and position part; 14 and 15 add the applications. The whole listing is given here once and referred to from the next two questions.

Steps

1. Copy the listing below into `scratch/adhoc-cbr.cc` inside your ns-3 directory (`cp adhoc-cbr.cc scratch/`).
2. Positions: a `ListPositionAllocator` receives one `Vector(x, y, z)` per node, node 0 at (0, 0, 1.5) m and node 1 at (distance, 0, 1.5) m. The z value is the antenna height, so both nodes have full 3D coordinates.
3. `MobilityHelper` takes that allocator, sets `ns3::ConstantPositionMobilityModel` (the nodes never move) and is installed on the `NodeContainer`.
4. Wi-Fi: `WifiHelper` with standard 802.11a and `ConstantRateWifiManager` fixed at `OfdmRate6Mbps`; `YansWifiChannelHelper::Default()` gives the log-distance loss model; `WifiMacHelper` type `ns3::AdhocWifiMac` makes it an ad-hoc network with no access point.
5. `InternetStackHelper` and `Ipv4AddressHelper` (10.1.1.0/24) give the two interfaces addresses 10.1.1.1 and 10.1.1.2.

6. Build and run: `./ns3 run "scratch/adhoc-cbr --distance=50"` . The first line printed comes from the mobility models and confirms the positions and the distance.

Program

adhoc-cbr.cc

```

1  /*
2  *  adhoc-cbr.cc  --  MCSL-223 Session 6, questions 13 to 15
3  *
4  *  Purpose of the program:
5  *    Two 802.11a ad-hoc nodes are placed at fixed 3D coordinates
6  *    (ConstantPositionMobilityModel fed by a ListPositionAllocator).
7  *    Node 0 runs a UdpClient (Q14) and a constant bit rate OnOff source (Q15)
8  *    node 1 runs the matching UdpServer and a PacketSink.  At the end the
9  *    program prints the positions, the distance, the packets the UdpServer
10 *    counted and the bytes the CBR sink delivered.
11 *
12 *  Topology:
13 *      n0 (0, 0, 1.5) m ~~~~ 802.11a ad-hoc, 6 Mbit/s ~~~~ n1 (d, 0, 1.5)
14 *      UdpClient  → port 9  → UdpServer
15 *      OnOff CBR  → port 10 → PacketSink
16 *
17 *  Build and run (ns-3.36 or later):
18 *    cp adhoc-cbr.cc scratch/
19 *    ./ns3 run "scratch/adhoc-cbr --distance=50"
20 *    ./ns3 run "scratch/adhoc-cbr --distance=100"
21 *    ./ns3 run "scratch/adhoc-cbr --distance=150"
22 */
23
24 #include "ns3/applications-module.h"
25 #include "ns3/core-module.h"
26 #include "ns3/internet-module.h"
27 #include "ns3/mobility-module.h"
28 #include "ns3/network-module.h"
29 #include "ns3/wifi-module.h"
30
31 using namespace ns3;
32
33 NS_LOG_COMPONENT_DEFINE("AdhocCbr");
34
35 /*
36 *  main: builds the two-node ad-hoc network, installs the applications,
37 *  runs for simTime seconds and prints the delivery figures.
38 */
39 int
40 main(int argc, char* argv[])
41 {
42     double distance = 50.0;      // metres between the nodes along x
43     double height = 1.5;        // antenna height, the z coordinate

```

```

44     double simTime = 10.0;           // seconds
45     std::string cbrRate = "500kb/s";
46     uint32_t cbrPacketSize = 500;    // bytes, so 500 kb/s is one packet every
47     bool verbose = false;
48
49     CommandLine cmd(__FILE__);
50     cmd.AddValue("distance", "Distance between the two nodes in metres", dis
51     cmd.AddValue("height", "Antenna height (z coordinate) in metres", height
52     cmd.AddValue("cbrRate", "Constant bit rate of the OnOff source", cbrRate
53     cmd.AddValue("simTime", "Simulation time in seconds", simTime);
54     cmd.AddValue("verbose", "Print UdpClient and UdpServer log lines", verbo
55     cmd.Parse(argc, argv);
56
57     if (verbose)
58     {
59         LogComponentEnable("UdpClient", LOG_LEVEL_INFO);
60         LogComponentEnable("UdpServer", LOG_LEVEL_INFO);
61     }
62
63     // ---- Q13: two nodes at fixed 3D positions -----
64     NodeContainer nodes;
65     nodes.Create(2);
66
67     Ptr<ListPositionAllocator> positions = CreateObject<ListPositionAllocato
68     positions->Add(Vector(0.0, 0.0, height));           // node 0
69     positions->Add(Vector(distance, 0.0, height));      // node 1
70
71     MobilityHelper mobility;
72     mobility.SetPositionAllocator(positions);
73     mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
74     mobility.Install(nodes);
75
76     // ---- Q13: 802.11a ad-hoc Wi-Fi at a fixed 6 Mbit/s -----
77     WifiHelper wifi;
78     wifi.SetStandard(WIFI_STANDARD_80211a);
79     wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",
80                                "DataMode", StringValue("OfdmRate6Mbps"),
81                                "ControlMode", StringValue("OfdmRate6Mbps"));
82
83     YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
84     YansWifiPhyHelper phy;
85     phy.SetChannel(channel.Create());
86
87     WifiMacHelper mac;

```

```
88     mac.SetType("ns3::AdhocWifiMac");
89
90     NetDeviceContainer devices = wifi.Install(phy, mac, nodes);
91
92     InternetStackHelper stack;
93     stack.Install(nodes);
94
95     Ipv4AddressHelper address;
96     address.SetBase("10.1.1.0", "255.255.255.0");
97     Ipv4InterfaceContainer ifaces = address.Assign(devices);
98
99     // ---- Q14: UdpServer on node 1, UdpClient on node 0 -----
100
101     uint16_t udpPort = 9;
102
103     uint32_t clientPackets = 900;
104
105     2
106
107     3     UdpServerHelper server(udpPort);
108
109     4     ApplicationContainer serverApp = server.Install(nodes.Get(1));
110
111     5     serverApp.Start(Seconds(0.0));
112
113     6     serverApp.Stop(Seconds(simTime));
114
115     7
116
117     8     UdpClientHelper client(ifaces.GetAddress(1), udpPort);
118
119     9     client.SetAttribute("MaxPackets", UintegerValue(clientPackets));
120
121     10     client.SetAttribute("Interval", TimeValue(MilliSeconds(10)));
122
123     11     client.SetAttribute("PacketSize", UintegerValue(1024));
124
125     12     ApplicationContainer clientApp = client.Install(nodes.Get(0));
126
127     13     clientApp.Start(Seconds(1.0));
128
129     14     clientApp.Stop(Seconds(simTime));
130
131     15
```

```
11
6    // ---- Q15: constant bit rate flow node 0 → node 1 -----
11
7    uint16_t cbrPort = 10;
11
8
11
9    OnOffHelper onoff("ns3::UdpSocketFactory",
12
10                        InetSocketAddress(Ifaces.GetAddress(1), cbrPort));
12
11    onoff.SetConstantRate(DataRate(cbrRate), cbrPacketSize); // OnTime 1, 0
12
12    ApplicationContainer cbrApp = onoff.Install(nodes.Get(0));
12
13    cbrApp.Start(Seconds(1.0));
12
14    cbrApp.Stop(Seconds(simTime));
12
15
12
16    PacketSinkHelper sinkHelper("ns3::UdpSocketFactory",
12
17                                    InetSocketAddress(Ipv4Address::GetAny(), cbr
12
18    ApplicationContainer sinkApp = sinkHelper.Install(nodes.Get(1));
12
19    sinkApp.Start(Seconds(0.0));
13
20    sinkApp.Stop(Seconds(simTime));
13
21
13
22    // pcap of node 1's Wi-Fi interface, for Wireshark
13
23    phy.EnablePcap("adhoc-cbr", devices.Get(1));
13
24
13
25    Simulator::Stop(Seconds(simTime));
13
26    Simulator::Run();
13
27
```

```

13
8    // ---- report -----
13
9    Ptr<MobilityModel> m0 = nodes.Get(0)→GetObject<MobilityModel>();
14
10   Ptr<MobilityModel> m1 = nodes.Get(1)→GetObject<MobilityModel>();
14
11   Vector p0 = m0→GetPosition();
14
12   Vector p1 = m1→GetPosition();
14
13
14   Ptr<UdpServer> udpServer = DynamicCast<UdpServer>(serverApp.Get(0));
14
15   Ptr<PacketSink> sink = DynamicCast<PacketSink>(sinkApp.Get(0));
14
16   double active = simTime - 1.0; // both sources run from 1 s to simTime
14
17
18   std::cout << "Node 0 at (" << p0.x << ", " << p0.y << ", " << p0.z << "
14
19       << "node 1 at (" << p1.x << ", " << p1.y << ", " << p1.z << "
15
20       << "distance " << m0→GetDistanceFrom(m1) << " m" << std::endl
15
21   std::cout << "UdpClient : sent " << clientPackets << " packets of 1024 b
15
22   std::cout << "UdpServer : received " << udpServer→GetReceived() << " pa
15
23       << udpServer→GetLost() << std::endl;
15
24   std::cout << "CBR sink : received " << sink→GetTotalRx() << " bytes in
15
25       << " s = " << sink→GetTotalRx() * 8.0 / active / 1000.0 << "
15
26       << std::endl;
15
27
15
28   Simulator::Destroy();

```

```

15
9     return 0;
16
0 }

```

Diagram

```

1      z = 1.5 m                                z = 1.5 m
2      n0 (0, 0, 1.5) ~~~~ 802.11a ad-hoc, 6 Mbit/s ~~~~ n1 (d, 0, 1.5)
3      10.1.1.1                                10.1.1.2
4      UdpClient ---- port 9 ----> UdpServer
5      OnOff CBR ---- port 10 ----> PacketSink

```

Output

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

```

1 $ ./ns3 run "scratch/adhoc-cbr --distance=50"
2 Node 0 at (0, 0, 1.5) m, node 1 at (50, 0, 1.5) m, distance 50 m
3 UdpClient : sent 900 packets of 1024 bytes
4 UdpServer : received 900 packets, lost 0
5 CBR sink  : received 562500 bytes in 9 s = 500 kbit/s

```

The file `adhoc-cbr-1-0.pcap` (node 1, device 0) is also written and opens in Wireshark.

Explanation

`GetDistanceFrom` computes the straight-line distance between the two position vectors:

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$$

With node 0 at (0, 0, 1.5) and node 1 at (50, 0, 1.5) the y and z terms cancel and d is 50 m. If you set the nodes at different heights the z term counts too, which is why the manual says "3D plane".

Ad-hoc mode means both stations talk directly; there is no association with an access point and no routing protocol is needed because both nodes are on the same subnet. Fixing the rate at 6

Mbit/s removes rate adaptation from the experiment, so the only thing that changes with distance is whether a frame is decoded or not.

Question 14

Problem Statement

■ Write in lab record

Install UDP server and Client at these two nodes.

Solution

■ Write in lab record

Steps

1. In the same script, `UdpServerHelper server(9)` installed on node 1 runs from 0 s to 10 s. `UdpServer` counts packets and, because the client adds a `SeqTsHeader`, also counts lost packets from gaps in the sequence numbers.
2. `UdpClientHelper client(10.1.1.2, 9)` installed on node 0 with attributes `MaxPackets` 900, `Interval` 10 ms and `PacketSize` 1024 bytes; it runs from 1 s to 10 s, so exactly 900 packets are offered.
3. Run with `--verbose` to see the per-packet log lines from both applications: `./ns3 run "scratch/adhoc-cbr --distance=50 --verbose=1"`.
4. Read `GetReceived()` and `GetLost()` on the `UdpServer` after `Simulator::Run()` and write both in the record.

Output

Expected log lines with `--verbose=1` (two of 900 shown; ns-3 prints times in nanoseconds):

```
1 TraceDelay TX 1024 bytes to 10.1.1.2 Uid: 4 Time: +1s
2 TraceDelay: RX 1024 bytes from 10.1.1.1 Sequence Number: 0 Uid: 4 TXtime: +1
3 TraceDelay TX 1024 bytes to 10.1.1.2 Uid: 6 Time: +1.01s
4 TraceDelay: RX 1024 bytes from 10.1.1.1 Sequence Number: 1 Uid: 6 TXtime: +1
5 ...
6 UdpServer : received 900 packets, lost 0
```

Explanation

The client offers 1024 bytes every 10 ms, which is

$$R = \frac{8 \times 1024}{0.010} = 819,200 \text{ bit/s}$$

well under the 6 Mbit/s link, so at 50 m nothing is lost. The 1.63 ms delay is almost all transmission time: 1024 bytes of payload plus 8 bytes UDP, 20 bytes IP, 8 bytes LLC/SNAP and 34 bytes MAC header and FCS is 1094 bytes, and

$$t_{\text{trans}} = \frac{1094 \times 8}{6 \times 10^6} = 1.46 \text{ ms}$$

plus the OFDM preamble (20 microseconds), DIFS and one backoff slot. Propagation over 50 m is 0.17 microseconds and does not show at this precision. `UdpServer` and `UdpClient` are the right pair here (rather than the echo pair from Session 1) because the server keeps the received and lost counters that the record needs.

Question 15

Problem Statement

■ Write in lab record

Setup a CBR transmission between these nodes.

Solution

■ Write in lab record

Steps

1. `OnOffHelper onoff("ns3::UdpSocketFactory", InetSocketAddress(10.1.1.2, 10))` on node 0. `SetConstantRate(DataRate("500kb/s"), 500)` sets `OnTime` to a constant 1, `OffTime` to a constant 0, `DataRate` 500 kb/s and `PacketSize` 500 bytes, which is exactly a constant bit rate source.
2. `PacketSinkHelper` on node 1 at port 10 receives it; `GetTotalRx()` gives the bytes delivered.
3. Both run from 1 s to 10 s. Run the script three times, at 50 m, 100 m and 150 m, and fill the table below.

4. Optional: `--cbrRate=1Mb/s` to see the effect of a heavier load.

Output

Expected values at the three distances (rows 2 and 3 depend on the random error model and will vary a little between runs; label them “approximate” in the record):

Distance	UdpServer received / sent	UdpServer lost	CBR bytes received	CBR throughput
50 m	900 / 900	0	562500	500 kbit/s
100 m	about 610 / 900	about 290	about 380000	about 338 kbit/s
150 m	0 / 900	900	0	0

```

1 $ ./ns3 run "scratch/adhoc-cbr --distance=150"
2 Node 0 at (0, 0, 1.5) m, node 1 at (150, 0, 1.5) m, distance 150 m
3 UdpClient : sent 900 packets of 1024 bytes
4 UdpServer : received 0 packets, lost 0
5 CBR sink : received 0 bytes in 9 s = 0 kbit/s

```

At 150 m `UdpServer` reports lost 0 because it never saw a single sequence number; the loss is the 900 packets that never arrived, which you compute yourself.

Explanation

Packet interval from the formula sheet:

$$\Delta t = \frac{8L}{R} = \frac{8 \times 500}{500,000} = 8 \text{ ms}$$

so in 9 s the source sends 1125 packets of 500 bytes, which is 562500 bytes, and the sink throughput is 8 times that over 9 s, exactly 500 kbit/s when nothing is lost.

Why distance matters: the default channel uses the log-distance propagation loss model with exponent 3 and a reference loss of 46.7 dB at 1 m, and the PHY transmits at 16 dBm. The received power is

$$P_r = P_t - L_0 - 10n \log_{10} d = 16.0 - 46.7 - 30 \log_{10} d \text{ dBm}$$

The noise floor of a 20 MHz 802.11a receiver with the default 7 dB noise figure is about -94 dBm, so the signal-to-noise ratio is

Distance	Received power	SNR	Result at 6 Mbit/s BPSK
50 m	-81.7 dBm	12.3 dB	every frame decoded
100 m	-90.7 dBm	3.3 dB	frame error rate large; part of the traffic lost
150 m	-96.0 dBm	below 0 dB	nothing decoded

Every 10 dB more path loss needs 2.15 times the distance with exponent 3, so the link goes from perfect to dead in a narrow band around 100 m. Put these three rows in the record next to the measured table.

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 2×10^8 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):

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

The maximum throughput of one TCP flow is bounded by

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

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:** Why `ConstantPositionMobilityModel` when nothing moves? **A:** Every Wi-Fi node needs a `MobilityModel` object so the channel can ask for its position; the constant model is the one that stores a fixed position and never updates it.
- **Q:** What does `ListPositionAllocator` do that `SetPositionAllocator` with a grid does not? **A:** It lets you give an explicit `Vector(x, y, z)` for each node in install order, so you control the exact 3D coordinates.
- **Q:** What makes the network ad-hoc? **A:** `WifiMacHelper::SetType("ns3::AdhocWifiMac")`. There is no access point and no association; frames go station to station.

- **Q:** Why fix the rate with `ConstantRateWifiManager` ? **A:** So the PHY rate does not adapt with distance. The measured loss then comes only from the propagation model, which is what the question studies.
- **Q:** How does an `OnOffHelper` become CBR? **A:** `SetConstantRate` sets `OnTime` to a constant 1 and `OffTime` to a constant 0, so the source is always on and sends one packet every $8L/R$ seconds.
- **Q:** Why does `UdpServer` report lost 0 at 150 m when nothing arrived? **A:** It counts gaps in the sequence numbers it sees; with no packets seen there are no gaps. The real loss is sent minus received.
- **Q:** Which model decides that 150 m is out of range? **A:** `LogDistancePropagationLossModel` with exponent 3; at 150 m the received power falls below the noise floor and the error-rate model rejects every frame.
- **Q:** Why is the delay about 1.6 ms for a 1024-byte packet? **A:** Transmission time at 6 Mbit/s for the 1094-byte frame (1.46 ms) plus preamble, DIFS and backoff; propagation over 50 m is negligible.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Leaving the default Wi-Fi standard (802.11ax in ns-3.36 and later) and asking for `OfdmRate6Mbps`; the run aborts because that mode does not exist for the standard. Call `SetStandard(WIFI_STANDARD_80211a)` first.
- Installing the mobility model after the Wi-Fi devices are used, or not at all; the channel then cannot compute a distance and the simulation asserts.
- Calling `GetReceived()` on the `ApplicationContainer` entry without `DynamicCast<UdpServer>`; the base `Application` class has no such method.
- Reading the CBR figure as packets per second instead of bytes; `GetTotalRx()` returns bytes, so multiply by 8 and divide by the active time to get bit/s.
- Running only one distance. The question is about the effect of distance; the record needs at least three rows.
- Forgetting that the client starts at 1 s while the server starts at 0 s, then dividing by 10 s instead of 9 s in the throughput.

Session Summary

■ Write in lab record

- Source listing of `adhoc-cbr.cc` with the header comment and the ASCII topology diagram (two nodes with their 3D coordinates)
- The distance formula with the two position vectors substituted
- Run output at 50 m with the `UdpServer` received and lost counts and the CBR bytes
- Two log lines from `--verbose=1` showing the UDP client TX and server RX with the delay
- The three-distance table (50, 100, 150 m) with received bytes, throughput and the received-power and SNR columns
- The 8 ms packet interval calculation and the 562500-byte check

[✎ Edit this page](#)

[< Previous](#)
Session 5

Session 7 [Next >](#)