

Session 4

Wireless ad-hoc network with OLSR

A mobile ad-hoc network has no access point; every node forwards for the others. OLSR is a proactive routing protocol that keeps routes ready before they are needed.

Objectives

✕ Do not copy. Read for understanding and the viva

- Complete questions 8 to 9 of the manual: wireless ad-hoc network with olsr
- 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
Q8	Take three nodes n1, n2 and n3 and create a wireless mobile ad-hoc network	Complete
Q9	Install the optimized Link State Routing protocol on these nodes	Complete

Preparation

✕ Do not copy. Read for understanding and the viva

- Wireless stack: WifiHelper, YansWifiPhyHelper, YansWifiChannelHelper, WifiMacHelper with AdhocWifiMac, and MobilityHelper with RandomWaypointMobilityModel for movement.
- Install OLSR with OlsrHelper passed to InternetStackHelper::SetRoutingHelper before installing the stack.

- Print routing tables at intervals with `Ipv4RoutingHelper::PrintRoutingTableAllAt` to show OLSR converging.

Question 8

Problem Statement

■ Write in lab record

Take three nodes n1, n2 and n3 and create a wireless mobile ad-hoc network.

Solution

■ Write in lab record

Questions 8 and 9 share one program, `adhoc_olsr.cc`. Question 8 is everything up to the mobility model; question 9 adds OLSR and the routing-table dumps. The code indices 0, 1, 2 are the manual's n1, n2, n3.

Steps

1. Save the program as `scratch/adhoc_olsr.cc` and run `./ns3 run scratch/adhoc_olsr`.
2. Read the `positions:` lines printed at 0, 10, 20 and 30 s; they change because the nodes move.
3. Draw the topology as three dots inside a 100 m by 100 m square with the starting positions from the 0 s line; mark the 11 Mbit/s 802.11b range (roughly 100 m with the default log-distance loss model).
4. For the record run again with `--pcap=true` and note the three files `adhoc-olsr-0-0.pcap`, `adhoc-olsr-1-0.pcap`, `adhoc-olsr-2-0.pcap`.

Program

adhoc_olsr.cc

```

1  /*
2   * MCSL-223 Section 1, Session 4, Questions 8 and 9
3   * Three-node mobile ad-hoc Wi-Fi network (RandomWaypoint) running OLSR.
4   *
5   * Build: copy to ns-3.36+/scratch/ and run
6   *   ./ns3 run scratch/adhoc_olsr
7   *
8   *   n1 (10.1.1.1)   n2 (10.1.1.2)   n3 (10.1.1.3)
9   *   802.11b ad-hoc, 11 Mbit/s, nodes move in a 100 m x 100 m box.
10  *   Node index 0, 1, 2 in the code = n1, n2, n3 in the manual.
11  *
12  *   Q8: nodes, Wi-Fi channel/phy/mac, mobility, IP stack, addresses.
13  *   Q9: OLSR through InternetStackHelper::SetRoutingHelper, routing
14  *       tables printed at 5 s, 15 s and 30 s, one UDP echo n1 → n3.
15  */
16  #include "ns3/core-module.h"
17  #include "ns3/network-module.h"
18  #include "ns3/internet-module.h"
19  #include "ns3/wifi-module.h"
20  #include "ns3/mobility-module.h"
21  #include "ns3/olsr-module.h"
22  #include "ns3/applications-module.h"
23
24  using namespace ns3;
25
26  NS_LOG_COMPONENT_DEFINE("AdhocOlsr");
27
28  /*
29   * PrintPositions: prints the current (x, y) of every node so the record
30   * shows the nodes really move. Reschedules itself every 10 s.
31   */
32  static void
33  PrintPositions(NodeContainer nodes)
34  {
35      std::cout << "t=" << Simulator::Now().GetSeconds() << "s positions:";
36      for (uint32_t i = 0; i < nodes.GetN(); ++i)
37      {
38          Vector p = nodes.Get(i)→GetObject<MobilityModel>()→GetPosition();
39          std::cout << "   n" << i + 1 << " (" << p.x << ", " << p.y << ")";
40      }
41      std::cout << std::endl;
42      Simulator::Schedule(Seconds(10.0), &PrintPositions, nodes);
43  }

```

```
44
45 /*
46  * main: builds the MANET (Q8), installs OLSR (Q9), schedules routing-table
47  * dumps and position prints, sends echo packets from n1 to n3 for 30 s.
48  */
49 int
50 main(int argc, char* argv[])
51 {
52     double simTime = 30.0;
53     bool pcap = false;
54
55     CommandLine cmd(__FILE__);
56     cmd.AddValue("simTime", "Simulation length in seconds", simTime);
57     cmd.AddValue("pcap", "Write per-node Wi-Fi pcap files", pcap);
58     cmd.Parse(argc, argv);
59
60     LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
61     LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
62
63     // ---- Q8: wireless mobile ad-hoc network ----
64     NodeContainer nodes;
65     nodes.Create(3);
66
67     // Physical layer and shared channel (log-distance propagation loss).
68     YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
69     YansWifiPhyHelper phy;
70     phy.SetChannel(channel.Create());
71
72     // 802.11b at a constant 11 Mbit/s so range is predictable (about 100 m)
73     WifiHelper wifi;
74     wifi.SetStandard(WIFI_STANDARD_80211b);
75     wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",
76                                 "DataMode", StringValue("DsssRate11Mbps"),
77                                 "ControlMode", StringValue("DsssRate11Mbps"));
78
79     // Ad-hoc MAC: no access point, every node is a peer.
80     WifiMacHelper mac;
81     mac.SetType("ns3::AdhocWifiMac");
82     NetDeviceContainer devices = wifi.Install(phy, mac, nodes);
83
84     // Mobility: RandomWaypoint inside a 100 m x 100 m box, 1 to 5 m/s, 2 s
85     ObjectFactory posFactory;
86     posFactory.SetTypeId("ns3::RandomRectanglePositionAllocator");
87     posFactory.Set("X", StringValue("ns3::UniformRandomVariable[Min=0.0|Max="
```

```

88 posFactory.Set("Y", StringValue("ns3::UniformRandomVariable[Min=0.0|Max=
89 Ptr<PositionAllocator> posAlloc = posFactory.Create()→GetObject<Positio
90
91 MobilityHelper mobility;
92 mobility.SetMobilityModel("ns3::RandomWaypointMobilityModel",
93     "Speed", StringValue("ns3::UniformRandomVariab
94     "Pause", StringValue("ns3::ConstantRandomVaria
95     "PositionAllocator", PointerValue(posAlloc));
96 mobility.SetPositionAllocator(posAlloc);
97 mobility.Install(nodes);
98
99 // ---- Q9: OLSR routing ----
100
101 OlsrHelper olsr;
102
103 Ipv4StaticRoutingHelper staticRouting;
104
105 Ipv4ListRoutingHelper list;
106
107 list.Add(staticRouting, 0);
108
109 list.Add(olsr, 10); // higher priority: OLSR is consulted first
110
111
112 InternetStackHelper stack;
113
114 stack.SetRoutingHelper(list); // must come before Install
115
116 stack.Install(nodes);
117
118
119 Ipv4AddressHelper address;
120
121 address.SetBase("10.1.1.0", "255.255.255.0");
122
123 Ipv4InterfaceContainer interfaces = address.Assign(devices);
124
125
126 // Routing tables of all nodes at 5 s, 15 s and 30 s.
127
128 Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper>(&st

```

```
11
6    olsr.PrintRoutingTableAllAt(Seconds(5.0), routingStream);
11
7    olsr.PrintRoutingTableAllAt(Seconds(15.0), routingStream);
11
8    olsr.PrintRoutingTableAllAt(Seconds(simTime), routingStream);
11
9
12
0    // Traffic to prove the routes work: n1 echoes to n3 every 2 s.
12
1    UdpEchoServerHelper echoServer(9);
12
2    ApplicationContainer serverApps = echoServer.Install(nodes.Get(2));
12
3    serverApps.Start(Seconds(1.0));
12
4    serverApps.Stop(Seconds(simTime));
12
5
12
6    UdpEchoClientHelper echoClient(interfaces.GetAddress(2), 9);
12
7    echoClient.SetAttribute("MaxPackets", UIntegerValue(10));
12
8    echoClient.SetAttribute("Interval", TimeValue(Seconds(2.0)));
12
9    echoClient.SetAttribute("PacketSize", UIntegerValue(512));
13
0    ApplicationContainer clientApps = echoClient.Install(nodes.Get(0));
13
1    clientApps.Start(Seconds(10.0)); // give OLSR time to converge
13
2    clientApps.Stop(Seconds(simTime));
13
3
13
4    if (pcap)
13
5    {
13
6        phy.EnablePcap("adhoc-olsr", devices);
13
7    }
```

```

13
8
13
9     Simulator::Schedule(Seconds(0.0), &PrintPositions, nodes);
14
0     Simulator::Stop(Seconds(simTime));
14
1     Simulator::Run();
14
2     Simulator::Destroy();
14
3     return 0;
14
4 }

```

Output

Expected position lines (the random stream is seeded to 1 by default, so a given NS-3 version prints the same numbers every run, but they differ between versions; the values here illustrate the format):

```

1 t=0s positions:  n1 (23.4, 71.2)  n2 (58.9, 44.7)  n3 (86.1, 12.5)
2 t=10s positions: n1 (41.6, 55.3)  n2 (61.2, 70.8)  n3 (70.4, 39.9)
3 t=20s positions: n1 (12.8, 30.1)  n2 (77.5, 81.6)  n3 (52.3, 63.0)
4 t=30s positions: n1 (33.0, 9.7)   n2 (90.2, 66.4)  n3 (48.8, 88.2)

```

Topology sketch for the record (positions at 0 s):

```

1  y
2  100 +-----+
3      |  n1 (23,71)      |
4      |                  |
5      |          n2 (59,45)      |  802.11b ad-hoc, 11 Mbit/s
6      |                  |  no access point, all peers
7      |                  n3      |
8  0  +-----+ x
9      0                  100

```

Explanation

The Wi-Fi stack, layer by layer:

Layer	Helper and type	Setting used here
Channel	<code>YansWifiChannelHelper::Default()</code>	constant speed propagation delay, log-distance path loss
Physical	<code>YansWifiPhyHelper</code>	default 802.11b transmit power, attached to the channel
Rate control	<code>ConstantRateWifiManager</code>	<code>DsssRate11Mbps</code> for data and control frames
MAC	<code>WifiMacHelper</code> type <code>ns3::AdhocWifiMac</code>	no association, no beacons
Standard	<code>WifiHelper::SetStandard</code>	<code>WIFI_STANDARD_80211b</code>
Mobility	<code>RandomWaypointMobilityModel</code>	speed 1 to 5 m/s, pause 2 s, 100 m by 100 m box

Five objects make a Wi-Fi node: a `YansWifiChannel` shared by all nodes with a propagation delay and a log-distance loss model, a `YansWifiPhy` per node attached to that channel, a `WifiMac` of type `AdhocWifiMac` (no association, no beacons from an access point, every node talks to every other directly), a `WifiNetDevice` that binds them, and a `MobilityModel` that gives the phy a position so the loss model can compute the received power. `ConstantRateWifiManager` pins the rate at 11 Mbit/s so the range is predictable. `RandomWaypointMobilityModel` picks a random destination inside the `RandomRectanglePositionAllocator`, moves there at a speed drawn from 1 to 5 m/s, pauses 2 s and repeats, which is why the positions differ at every print. Mobile means the neighbour set changes over time; that is what makes a routing protocol necessary in question 9.

Question 9

Problem Statement

■ Write in lab record

Install the optimized Link State Routing protocol on these nodes.

Solution

■ Write in lab record

Steps

1. Create an `OlsrHelper` and an `Ipv4StaticRoutingHelper`, put both in an `Ipv4ListRoutingHelper` (static at priority 0, OLSR at priority 10).
2. Call `InternetStackHelper::SetRoutingHelper(list)` before `Install`; the order matters because routing is chosen when the stack is built.
3. Schedule `PrintRoutingTableAllAt` at 5 s, 15 s and 30 s so the record shows the tables converging and changing as nodes move.
4. Start the echo client at 10 s, after OLSR has exchanged HELLO (every 2 s) and TC (every 5 s) messages and filled the tables.
5. Run and copy the three routing-table dumps and the echo lines into the record.

Program

Same file as question 8; the OLSR part is:

CPP

```
1 OlsrHelper olsr;
2 Ipv4StaticRoutingHelper staticRouting;
3 Ipv4ListRoutingHelper list;
4 list.Add(staticRouting, 0);
5 list.Add(olsr, 10);
6
7 InternetStackHelper stack;
8 stack.SetRoutingHelper(list);
9 stack.Install(nodes);
10
11 Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper>(&std::c
12 olsr.PrintRoutingTableAllAt(Seconds(5.0), routingStream);
```

Output

Expected routing table dump at 5 s for n1 (node 0). The OLSR table lists every other node with its next hop and hop count; here n3 is two hops away through n2. The same block repeats for nodes

1 and 2 and again at 15 s and 30 s with different next hops as the nodes move.

```

1 Node: 0, Time: +5s, Local time: +5s, Ipv4ListRouting table
2   Priority: 10 Protocol: ns3::olsr::RoutingProtocol
3 Node: 0, Time: +5s, Local time: +5s, OLSR Routing table
4 Destination      NextHop      Interface      Distance
5 10.1.1.2          10.1.1.2      1              1
6 10.1.1.3          10.1.1.2      1              2
7
8   Priority: 0 Protocol: ns3::Ipv4StaticRouting
9 Node: 0, Time: +5s, Local time: +5s, Ipv4StaticRouting table
10 Destination      Gateway      Genmask         Flags Metric Ref    Use Ifac
11 127.0.0.0         0.0.0.0      255.0.0.0       U      0      -     -     0
12 10.1.1.0          0.0.0.0      255.255.255.0   U      0      -     -     1

```

Expected echo lines (n1 to n3, 512 bytes, every 2 s from 10 s; the delay is about 1.5 ms per Wi-Fi hop at 11 Mbit/s, so a two-hop path shows about 3 ms each way):

```

1 At time +10s client sent 512 bytes to 10.1.1.3 port 9
2 At time +10.003s server received 512 bytes from 10.1.1.1 port 49153
3 At time +10.003s server sent 512 bytes to 10.1.1.1 port 49153
4 At time +10.006s client received 512 bytes from 10.1.1.3 port 9
5 At time +12s client sent 512 bytes to 10.1.1.3 port 9
6 At time +12.0015s server received 512 bytes from 10.1.1.1 port 49153
7 At time +12.0015s server sent 512 bytes to 10.1.1.1 port 49153
8 At time +12.003s client received 512 bytes from 10.1.1.3 port 9
9 ...

```

Per-hop delay from the formula sheet for a 512-byte echo at 11 Mbit/s: frame on air is $512 + 8 + 20 + 8$ (LLC) + 24 (MAC) + 4 (FCS) = 576 bytes, $t_{\text{trans}} = 576 \times 8 / 11 \times 10^6 = 0.42$ ms, plus 0.192 ms PLCP preamble and header, 0.05 ms DIFS, about 0.3 ms average backoff and 0.3 ms for the MAC ACK: about 1.3 to 1.5 ms per hop. Propagation over 100 m is 0.33 microseconds and can be ignored.

Explanation

OLSR (RFC 3626) is proactive: every node broadcasts HELLO messages to learn its one-hop and two-hop neighbours, elects multipoint relays (MPRs) that are the only nodes to forward its topology control (TC) messages, and runs Dijkstra on the resulting link-state graph. Routes

therefore exist before any data is sent, which is why the tables at 5 s are already complete and the echo at 10 s succeeds without a route-discovery delay. When a node moves out of range, missing HELLOs expire the link after the neighbour hold time (6 s by default) and the table is recomputed; the dumps at 15 s and 30 s show the next hop for 10.1.1.3 changing between direct and via 10.1.1.2. `Ipv4ListRouting` tries protocols in priority order, so OLSR answers first and static routing only handles the loopback and the local subnet.

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** What makes a network ad-hoc? **A:** No access point; every node forwards for others using a routing protocol.
- **Q:** What does `RandomWaypointMobilityModel` need that other models do not? **A:** A `PositionAllocator` attribute to draw its waypoints from.
- **Q:** Why must `SetRoutingHelper` come before `Install`? **A:** The stack helper creates the routing protocol object while installing; changing it later has no effect on nodes already built.
- **Q:** Proactive or reactive: which is OLSR, and what is the trade-off? **A:** Proactive; routes are ready before use, at the cost of periodic HELLO and TC traffic even when idle.
- **Q:** What is a multipoint relay? **A:** A neighbour chosen to reforward TC messages so that flooding reaches every two-hop neighbour with fewer transmissions.
- **Q:** Why is the echo client started at 10 s and not 1 s? **A:** OLSR needs a few HELLO and TC intervals to build the tables; early packets would be dropped for lack of a route.
- **Q:** What does Distance 2 mean in the OLSR table? **A:** The destination is two hops away; the packet goes to NextHop first.
- **Q:** How is the radio range set in this script? **A:** Indirectly: transmit power, the log-distance loss model and the 11 Mbit/s receive threshold give roughly 100 m.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Using `StaWifiMac` or `ApWifiMac`; an ad-hoc network needs `AdhocWifiMac`.
- Forgetting the `PositionAllocator` attribute on `RandomWaypointMobilityModel`, which aborts the run.

- Installing the internet stack before calling `SetRoutingHelper`, so nodes get global routing instead of OLSR and the echo fails when nodes are more than one hop apart.
- Sending traffic at 1 s and reporting that OLSR does not work.
- Printing only the static routing table and missing the OLSR block above it.
- Making the box much larger than the radio range with only three nodes, so the network is partitioned most of the time.

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):

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

The maximum throughput of one TCP flow is bounded by

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

Packet loss

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

Session Summary

Write in lab record

- `adhoc_olsr.cc` listing with the Wi-Fi, mobility and OLSR parts marked as questions 8 and 9
- Position printout at 0, 10, 20, 30 s and the topology sketch inside the 100 m box
- OLSR routing tables of all three nodes at 5, 15 and 30 s, and the echo lines with the per-hop delay estimate

[Edit this page](#)

< Previous
Session 3

Next >
Session 5