无线宽带自组网通信系统

OLSR 协议仿真实验报告

部　　门：研发部通信项目组

姓　　名：罗敏

版　　本：1.1

使用 LaTeX 撰写于 2018 年 6 月 6 日

摘　要

无人机集群是一个新的应用领域，特别是在军事领域, 将会引发一场革命性的变革，而多机之间的通信又是无人机集群飞行之中的关键，如何实现更好的无几人集群之间的组网是一个需要深入研究和实验的方向。因应用领域的特殊性，真实环境采集数据测试具有很大的困难且花费大，因而需要预先实验仿真，本文档主要是仿真无人机集群使用 olsr 协议进行自组网通信, 并评估 olsr 协议在组网过程中的各项性能指标和参数，为后续改进协议做参考。

关键字：　仿真　无人机集群　olsr　性能

目录

实验一　路由生成与切换

1.1　实验目的

本项实验主要目的有两个, 具体如下：

- 评估 OLSR 协议在整个无人机集群组网过程中路由生成时间
- 评估 OLSR 协议拓扑变化时路由切换时间。

1.2　实验工具

本实验采用 NS3 仿真评估软件作为仿真工具，NS3 平台提供了一套完成的协议仿真与数据记录工具，能够有效的仿真协议行为并在接近理想环境下评估协议性能, 需要了解更多详细信息可以参考 NS3 官方网站[1]。

模拟节点运动的工具采用的是 NetAnim, 可以模拟节点运行过程中的行为和链路的通断与连接。

1.3　实验设计

根据实验目的，本实验采用如图 1.1 所示实验拓扑，每架无人机之间设置间隔距离 500m，1 号、2 号和 3 号飞机相对静止，0 号飞机进入飞行集群编队，速度为 20m/s, 整个仿真过程 120s，实时记录各个点之间通信报文并且每 0.2 秒打印一次每个节点的路由表变化情况，通过分析路由表变化记录可以得到路由切换数据和路由生成数据。0 号节点飞行的过程中一次会和 1 号、2 号、3 好节点相连接，路由也会有一个依次切换的过程。

1.4　数据分析

为了便于比较 hello 报文和 tc 报文对路由收敛的影响，本实验分别采集了 hello 报文和 tc 四组不同的设置值时路由表的收敛变化情况，具体如图 1.2 所示，在图中子图 (a) 是 hello

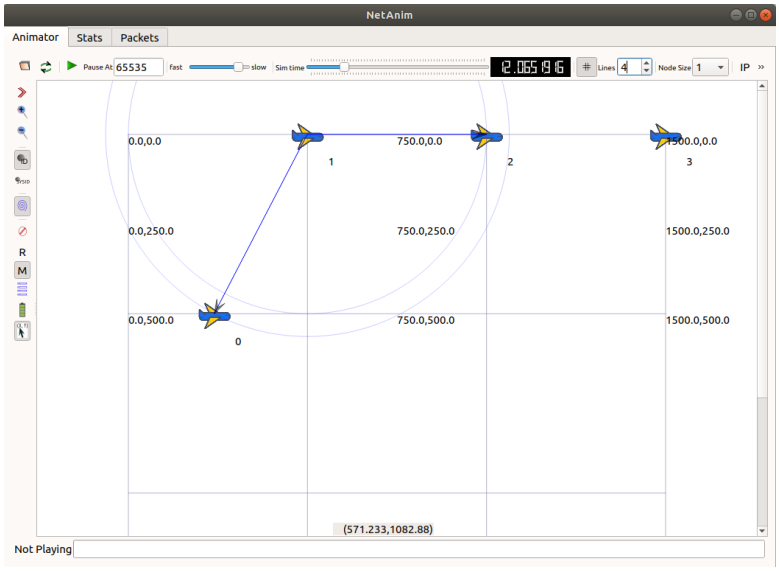
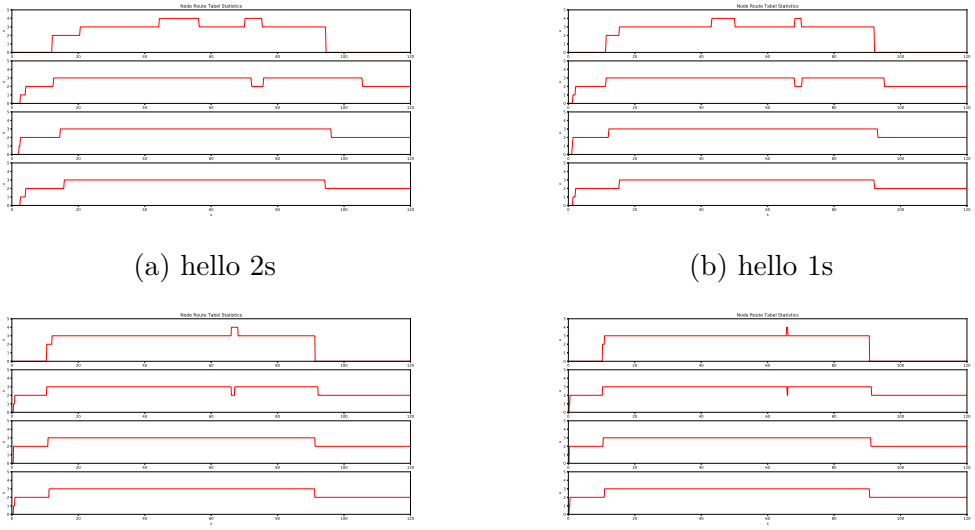


图 1.1: 仿真实验拓扑

报文设置为 2s，tc 报文设置为 5s 时的数据采集图，子图 (b)(c)(d) 对应的设置为 hello 报文为 1s、0.5s 和 0.2s，对应的 tc 报文为 2.5s、1s 和 0.4s。



(a) hello 2s

(b) hello 1s

(c) hello 0.4s

(d) hello 0.2s

图 1.2: 路由数据统计

根据子图 (a) 中仿真采集的路由表数据，可以很直接看到图中依次是 node0 到 node3 四个节点的路由表条目统计，又图中的路由项的变化可以很直观的看到路由逐条增加，即节点之间开始组网，最开始是 1 号、2 号、3 号节点相互组网，时间基本是一致的，都是在 2.6s 左右的时间开始形成路由，但是图中还是有路由异常变动的地方, 对应的异常变动点后面会依次进行分析。

```
11968  Node: 0, Time: +44.19999999999999999999s, Local time: +44.19999999999999999999s, OLSR Routing table
11969  Destination     NextHop     Interface   Distance
11970  10.1.1.2        10.1.1.2        1       1
11971  10.1.1.3        10.1.1.3        1       1
11972  10.1.1.4        10.1.1.3        1       2
12024  Node: 0, Time: +44.39999999999999999998s, Local time: +44.39999999999999999998s, OLSR Routing table
12025  Destination     NextHop     Interface   Distance
12026  10.1.1.1        10.1.1.3        1       3
12027  10.1.1.2        10.1.1.3        1       2
12028  10.1.1.3        10.1.1.3        1       1
12029  10.1.1.4        10.1.1.3        1       2
```

图 1.3: 节点 0 异常变动点

可以先来分析一下节点 0 出现的异常情况，针对异常变动点的时间找到了原文件中的路由变化如图 1.3 所示, 在路由记录文件中可以看到节点 0 居然多出了一条路由, 这条路由是节点 0 自己的路由，这应该是出现 bug 或者记录出错了，形成了环回路由, 这应该是错误的。

```
19834  Node: 1, Time: +72.0s, Local time: +72.0s, Ipv4ListRouting table
19835    Priority: 10 Protocol: ns3::olsr::RoutingProtocol
19836  Node: 1, Time: +72.0s, Local time: +72.0s, OLSR Routing table
19837  Destination     NextHop     Interface   Distance
19838  10.1.1.1        10.1.1.3        1       2
19839  10.1.1.3        10.1.1.3        1       1
19840  10.1.1.4        10.1.1.3        1       2
20898  Node: 1, Time: +75.799999999999999996s, Local time:
       +75.799999999999999996s, OLSR Routing table
20899  Destination     NextHop     Interface   Distance
20900  10.1.1.1        10.1.1.3        1       3
20901  10.1.1.3        10.1.1.3        1       1
20902  10.1.1.4        10.1.1.3        1       2
```

图 1.4: 节点 1 路由变化

针对 (a) 子图异常点找到了对应的找到了节点 1 此时的路由，对比如图 1.4 所示。个点的路由变化从截图很容易看出来，1 号点从与 0 好点直连切换到了与 0 号点的通信需要 2 号点做中继, 但是这个路由切换非常迅速，在实验中采用的是 0.2s 打印一次路由表，对比结果可以看出来，在 0.2s 内路由便实现了切换, 这个切换时间是很快的，已经达到了毫秒级。

在另外的一个路由切换点则路由切换时间要长的多，根据图 1.5 可以得到，路由切换时间达到了 3.7s。同时可以对比 (b)(c)(d) 图，随着 hello 报文和 tc 报文间隔的缩小，路由切换时间也在变短。

根据节点运行轨迹，对应画了一下拓扑的变化情况，如图 1.5 所示, 节点拓扑变化非常简单，做了两次路由切换，这期间发生了两次的路由切换，第一次切换时间非常短，但是第二

```
11982  Node: 1, Time: +44.19999999999999999999s, Local time: +44.19999999999999999999s, OLSR Routing table
11983  Destination     NextHop     Interface   Distance
11984  10.1.1.1        10.1.1.1        1       1
11985  10.1.1.3        10.1.1.3        1       1
11986  10.1.1.4        10.1.1.3        1       2
12039  Node: 1, Time: +44.39999999999999999998s, Local time: +44.39999999999999999998s, OLSR Routing table
12040  Destination     NextHop     Interface   Distance
12041  10.1.1.1        10.1.1.3        1       2
12042  10.1.1.3        10.1.1.3        1       1
12043  10.1.1.4        10.1.1.3        1       2
```

图 1.5: 节点 1 路由变化

次路由切换的时候则出现了长时间路由未收敛的情况，根据参考 OLSR 协议的 RFC 文档[2], 在此处应该出现了 MPRs 的重新选举, 从而导致对整个拓扑路由的重新计算。

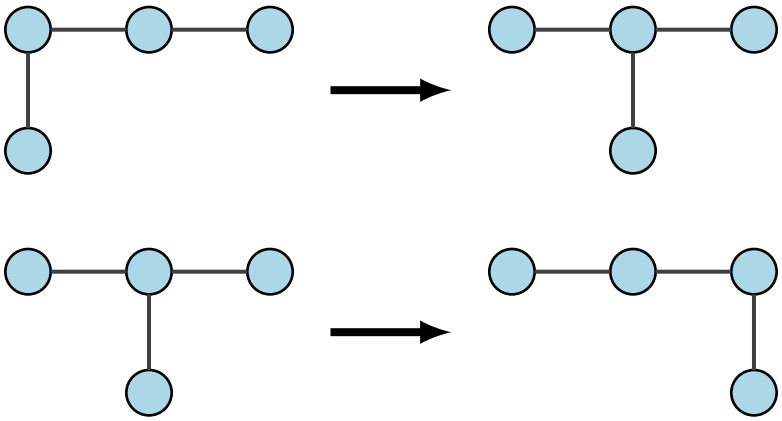


图 1.6: 拓扑变化图

1.5 实验结论

本实验的结论：协议路由切换时间与 hello 报文和 tc 报文正相关，hello 报文和 tc 报文设置的越小，路由表更新时间越快。

实验二　路由切换与数据传输

2.1　实验目的

本实验的目的是验证 olsr 协议报文时钟与网络中报文数据量之间的关系。

2.2　实验工具

同上一实验。

2.3　实验设计

实验拓扑仍然采用图 1.1 所示，通过设置不同的 hello 和 tc 报文时钟，然后计算整个拓扑中 OLSR 协议的报文流通量再除以整个测试时间计算式如下：

公式 1 (OLSR 协议流量计算)

数学表达式：

$$(\sum_{i=1}^{n} X_i)/t \xrightarrow{\text{p.s.}} \mathbb{E}(X_i).$$

其中 Xi 表示 i 节点发出的报文总数,t 表示拓扑测试时间。

根据公式, 分别设置了四组 hello 和 tc 报文时钟，分别是（2.0, 5.0）,(1.0, 2.5), (0.5, 1), (0.2, 0.4), 然后运行过程中使用 wireshark 统计了各个节点的数据流量，记录了拓扑测试时间。

2.4　数据分析

在图 2.1 中记录了四组时钟情况下的流通量，根据图可以看到，随着 hello 报文和 tc 报文时钟的变短，报文量快速上升，基本上呈现一个线性上升的关系, 而且上升趋势比较平缓，处于可以接受的范围内。

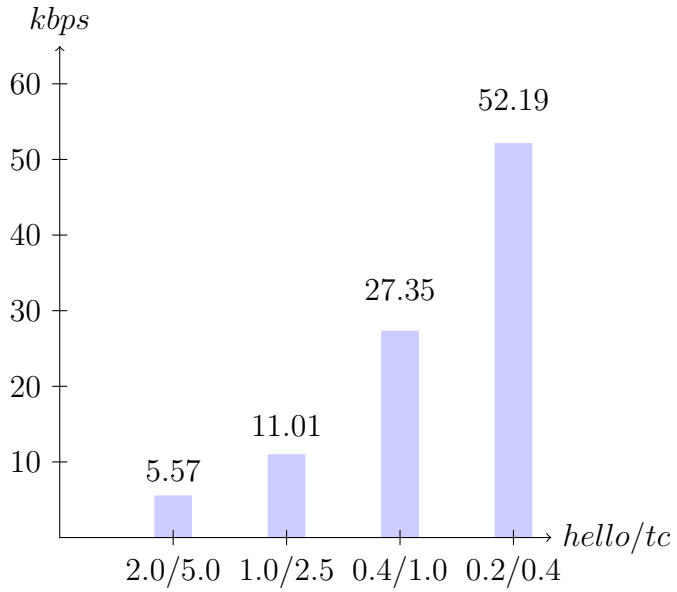


图 2.1: 报文数据流量

同时我们关心单位时间内报文流量的情况在当 hello 和 tc 报文分别设置到 (0.2, 0.4), 根据图 1.2 的 (d) 子图可以看到，路由切换基本能够满足在 300ms 左右完成切换, 且此时流量维持在 52kbps 左右，这样的流量大小和切换时延是能够接受的。

2.5 实验结论

本实验结论: 一定程度上缩短 hello 报文和 tc 报文的时钟，可以加速路由表的切换，且拓扑中所增加的报文流量是能够接受和可以预期的。

实验三　模拟无人机编队飞行

3.1　实验目的

本实验主要是在多架次无人机编队飞行过程中，观察 olsr 协议的运行情况和不同报文时钟的情况下协议运行效率的变化。

3.2　实验工具

同上

3.3　实验设计

将 20 架次的无人机进行 5×4 编队，中心站点则在中间，无人机编队设置 4 个航点，围绕中心站飞行，如图 3.1 所示。

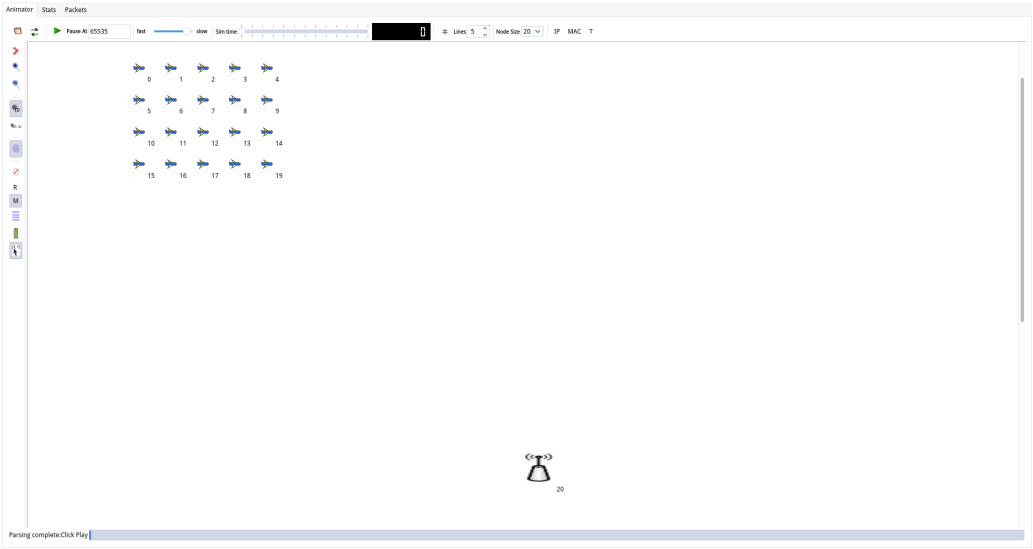


图 3.1: 模拟无人机集群飞行

因无人机集群是以一个密集编队飞行，在仿真实验中，设置无人机之间相互距离为 50 米，无人机集群距离中心站点是 1000 米的距离，在这样的飞行模型下，无人机均可以与中心站点

相互连接，并不需要进行中继。本实验主要研究在改变报文定时器的情况下多机集群协议报文的流量，根据实验目的设计了五组数据进行分析。

表 3.1: 测试组

报文组	hello 报文 (s)	tc 报文 (s)
第一组	2	5
第二组	1	2.5
第三组	0.4	1
第四组	0.2	0.5
第五组	0.1	0.25

3.4 数据分析

通过实验模拟之后收集各个节点报文流量数据再经过分析得到图 3.2 的统计表, 根据统计可以看到在 21 节点情况下每秒钟全拓扑协议报文数据量相比 4 个节点点有了将近 10-11 倍左右的上升，这个上升幅度还是比较大的，当然，同时也要考虑到 21 节点情况属于密集组网的飞行模型，所有节点都是互联互通，这种情况下报文流量是最大的。

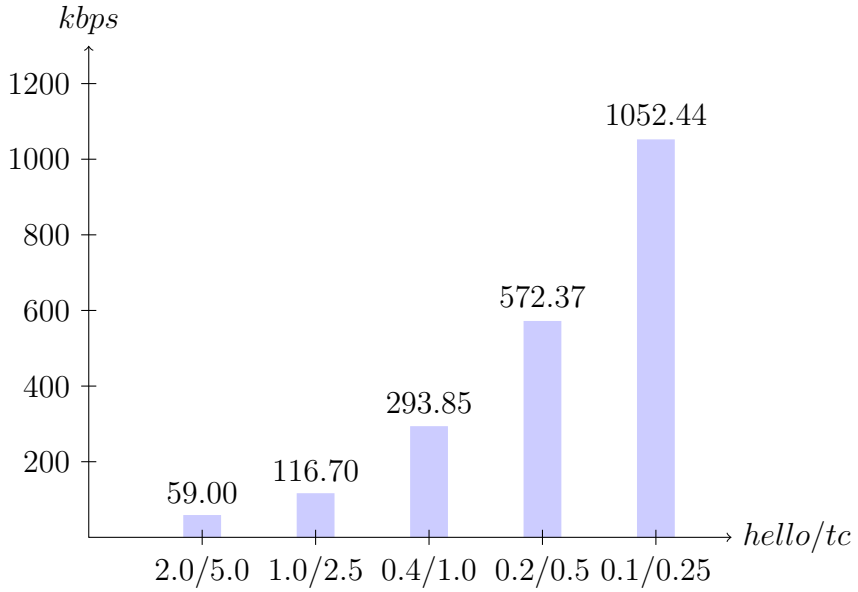


图 3.2: 报文数据流量

同时，在相应改变 hello/tc 报文时钟的情况下，报文流量也出现了一定幅度的上升，基本是按一个线性比例的关系上升的，比例在 1:1.9 左右, 即报文时间调整一倍，报文流量接近翻一倍。可以看到在将 hello/tc 报文调整到 0.1s 和 0.25s 的情况下，整个拓扑里面的流量还

是比较大的，达到了 1052.44kbps，这个流量已经很大了，所以在进行报文时钟调整时要根据实际情况来进行配置，兼顾路由切换的时间要求和协议报文对整个链路的流量占比。

3.5 实验结论

根据模拟集群飞行试验，可以得到了一个报文流量关系的比例，以及基本可以判断在不同的报文时钟配置情况下，整个拓扑中的协议报文数量，这样的结果是比较有参考意义的，可以知道在实际的布点的过程中设置一个合适的报文时钟。

实验总结

通过本次仿真实验加深了对于 OLSR 协议的理解，也从理论上指导了在设备上调试运行 OLSR 协议，并且为后续针对 OLSR 协议的改进打下了良好的基础。

参 考 文 献

[1] https://www.nsnam.org/

[2] Network Working Group. OLSR: Optimized Link State Routing Protocol, 2003.

附录 A 数据表

1. 路由表记录文件

2. 各节点抓包文件

附录 B 程序代码

下面是 ns3 仿真的 C++ 程序，分别设置了各个节点的属性和测试过程, 具体可以参考代码内容。

```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 University of Washington
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  ...
    USA
 *
 */

//
// This program configures a grid (default 5x5) of nodes on an
// 802.11b physical layer, with
// 802.11b NICs in adhoc mode, and by default, sends one packet of 1000
// (application) bytes to node 1.
//
// The default layout is like this, on a 2-D grid.
//
// n20  n21  n22  n23  n24
// n15  n16  n17  n18  n19
// n10  n11  n12  n13  n14
```

```
31 // n5   n6   n7   n8   n9
32 // n0   n1   n2   n3   n4
33 //
34 // the layout is affected by the parameters given to GridPositionAllocator;
35 // by default, GridWidth is 5 and numNodes is 25..
36 //
37 // There are a number of command-line options available to control
38 // the default behavior.  The list of available command-line options
39 // can be listed with the following command:
40 // ./waf --run "wifi-simple-adhoc-grid --help"
41 //
42 // Note that all ns-3 attributes (not just the ones exposed in the below
43 // script) can be changed at command line; see the ns-3 documentation.
44 //
45 // For instance, for this configuration, the physical layer will
46 // stop successfully receiving packets when distance increases beyond
47 // the default of 500m.
48 // To see this effect, try running:
49 //
50 // ./waf --run "wifi-simple-adhoc --distance=500"
51 // ./waf --run "wifi-simple-adhoc --distance=1000"
52 // ./waf --run "wifi-simple-adhoc --distance=1500"
53 //
54 // The source node and sink node can be changed like this:
55 //
56 // ./waf --run "wifi-simple-adhoc --sourceNode=20 --sinkNode=10"
57 //
58 // This script can also be helpful to put the Wifi layer into verbose
59 // logging mode; this command will turn on all wifi logging:
60 //
61 // ./waf --run "wifi-simple-adhoc-grid --verbose=1"
62 //
63 // By default, trace file writing is off-- to enable it, try:
64 // ./waf --run "wifi-simple-adhoc-grid --tracing=1"
65 //
66 // When you are done tracing, you will notice many pcap trace files
67 // in your directory.  If you have tcpdump installed, you can try this:
```

```
68 //
69 // tcpdump -r wifi-simple-adhoc-grid-0-0.pcap -nn -tt
70 //
71
72 #include "ns3/core-module.h"
73 #include "ns3/mobility-module.h"
74 #include "ns3/wifi-module.h"
75 #include "ns3/internet-module.h"
76 #include "ns3/olsr-helper.h"
77
78 #include "ns3/netanim-module.h"
79
80 using namespace ns3;
81
82 NS_LOG_COMPONENT_DEFINE ("WifiSimpleAdhocGrid");
83
84 void ReceivePacket (Ptr<Socket> socket)
85 {
86     while (socket->Recv ())
87     {
88         NS_LOG_UNCOND ("Received one packet!");
89     }
90 }
91
92 static void GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize,
93                              uint32_t pktCount, Time pktInterval )
94 {
95     if (pktCount > 0)
96     {
97         socket->Send (Create<Packet> (pktSize));
98         Simulator::Schedule (pktInterval, &GenerateTraffic,
99                              socket, pktSize,pktCount - 1, pktInterval);
100     }
101     else
102     {
103         socket->Close ();
104     }
```

```
105 }
106
107
108 int main (int argc, char *argv[])
109 {
110     std::string phyMode ("DsssRate1Mbps");
111     double distance = 500;  // m
112     uint32_t packetSize = 1000; // bytes
113     uint32_t numPackets = 100;
114     uint32_t numNodes = 4;  // by default, 5x5
115     uint32_t sinkNode = 0;
116     uint32_t sourceNode = 3;
117     double interval = 1.0; // seconds
118     bool verbose = false;
119     bool tracing = false;
120     double HelloInterval = 2.0;
121     double TcInterval = 5.0;
122     double MidInterval = 5.0;
123     double HnaInterval = 5.0;
124
125     CommandLine cmd;
126
127     cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
128     cmd.AddValue ("distance", "distance (m)", distance);
129     cmd.AddValue ("packetSize", "size of application packet sent", ...
        packetSize);
130     cmd.AddValue ("numPackets", "number of packets generated", numPackets);
131     cmd.AddValue ("interval", "interval (seconds) between packets", ...
        interval);
132     cmd.AddValue ("verbose", "turn on all WifiNetDevice log components", ...
        verbose);
133     cmd.AddValue ("tracing", "turn on ascii and pcap tracing", tracing);
134     cmd.AddValue ("numNodes", "number of nodes", numNodes);
135     cmd.AddValue ("sinkNode", "Receiver node number", sinkNode);
136     cmd.AddValue ("sourceNode", "Sender node number", sourceNode);
137     cmd.AddValue ("hellointerval", "OLSR Routing Protocol hello packet ...
        interval", HelloInterval);
```

```
138     cmd.AddValue ("tcinterval", "OLSR Routing Protocol Tc packet interval", ...
            TcInterval);
139     cmd.AddValue ("midinterval", "OLSR Routing Protocol Mid interval", ...
            MidInterval);
140     cmd.AddValue ("hnainterval", "OLSR Routing Protocol Hna interval", ...
            HnaInterval);
141
142     cmd.Parse (argc, argv);
143     // Convert to time object
144     Time interPacketInterval = Seconds (interval);
145
146     // disable fragmentation for frames below 2200 bytes
147     Config::SetDefault ...
            ("ns3::WifiRemoteStationManager::FragmentationThreshold", ...
            StringValue ("2200"));
148     // turn off RTS/CTS for frames below 2200 bytes
149     Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", ...
            StringValue ("2200"));
150     // Fix non-unicast data rate to be the same as that of unicast
151     Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
152                         StringValue (phyMode));
153
154     NodeContainer c;
155     c.Create (numNodes);
156
157     // The below set of helpers will help us to put together the wifi NICs ...
            we want
158     WifiHelper wifi;
159     if (verbose)
160     {
161       wifi.EnableLogComponents ();  // Turn on all Wifi logging
162     }
163
164     YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
165     // set it to zero; otherwise, gain will be added
166     wifiPhy.Set ("RxGain", DoubleValue (-10) );
167     // ns-3 supports RadioTap and Prism tracing extensions for 802.11b
```

```
168     wifiPhy.SetPcapDataLinkType (WifiPhyHelper::DLT_IEEE802_11_RADIO);
169
170     YansWifiChannelHelper wifiChannel;
171     wifiChannel.SetPropagationDelay ...
            ("ns3::ConstantSpeedPropagationDelayModel");
172     wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
173     wifiPhy.SetChannel (wifiChannel.Create ());
174
175     // Add an upper mac and disable rate control
176     WifiMacHelper wifiMac;
177     wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
178     wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
179                                   "DataMode",StringValue (phyMode),
180                                   "ControlMode",StringValue (phyMode));
181     // Set it to adhoc mode
182     wifiMac.SetType ("ns3::AdhocWifiMac");
183     NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);
184
185     MobilityHelper mobility;
186     mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
187                                    "MinX", DoubleValue (0.0),
188                                    "MinY", DoubleValue (0.0),
189                                    "DeltaX", DoubleValue (distance),
190                                    "DeltaY", DoubleValue (distance),
191                                    "GridWidth", UintegerValue (4),
192                                    "LayoutType", StringValue ("RowFirst"));
193     //mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
194     mobility.SetMobilityModel ("ns3::ConstantVelocityMobilityModel");
195     mobility.Install (c);
196
197     c.Get (0)->GetObject<MobilityModel> ()->SetPosition (Vector (0, 500, 0));
198     c.Get (0)->GetObject<ConstantVelocityMobilityModel> ()->SetVelocity ...
            (Vector (20, 0, 0));
199
200     // Enable OLSR
201     OlsrHelper olsr;
202     Ipv4StaticRoutingHelper staticRouting;
```

```
203
204     Time t_HelloInterval = Seconds (HelloInterval);
205     olsr.Set("HelloInterval", TimeValue(t_HelloInterval));
206
207     Time t_TcInterval = Seconds (TcInterval);
208     olsr.Set("TcInterval", TimeValue(t_TcInterval));
209
210     Time t_MidInterval = Seconds (MidInterval);
211     olsr.Set("MidInterval", TimeValue(t_MidInterval));
212
213     Time t_HnaInterval = Seconds (HnaInterval);
214     olsr.Set("HnaInterval", TimeValue(t_HnaInterval));
215
216
217     Ipv4ListRoutingHelper list;
218     list.Add (staticRouting, 0);
219     list.Add (olsr, 10);
220
221     InternetStackHelper internet;
222     internet.SetRoutingHelper (list); // has effect on the next Install ()
223     internet.Install (c);
224
225     Ipv4AddressHelper ipv4;
226     NS_LOG_INFO ("Assign IP Addresses.");
227     ipv4.SetBase ("10.1.1.0", "255.255.255.0");
228     Ipv4InterfaceContainer i = ipv4.Assign (devices);
229
230     TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
231     Ptr<Socket> recvSink = Socket::CreateSocket (c.Get (sinkNode), tid);
232     InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);
233     recvSink->Bind (local);
234     recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));
235
236     Ptr<Socket> source = Socket::CreateSocket (c.Get (sourceNode), tid);
237     InetSocketAddress remote = InetSocketAddress (i.GetAddress (sinkNode, ...
            0), 80);
238     source->Connect (remote);
```

```
239
240     if (tracing == true)
241     {
242         AsciiTraceHelper ascii;
243         wifiPhy.EnableAsciiAll (ascii.CreateFileStream ...
                ("wifi-simple-adhoc-grid.tr"));
244         wifiPhy.EnablePcap ("wifi-simple-adhoc-grid", devices);
245         // Trace routing tables
246         Ptr<OutputStreamWrapper> routingStream = ...
                Create<OutputStreamWrapper> ("wifi-simple-adhoc-grid.routes", ...
                std::ios::out);
247         olsr.PrintRoutingTableAllEvery (Seconds (0.1), routingStream);
248         Ptr<OutputStreamWrapper> neighborStream = ...
                Create<OutputStreamWrapper> ("wifi-simple-adhoc-grid.neighbors", ...
                std::ios::out);
249         olsr.PrintNeighborCacheAllEvery (Seconds (0.1), neighborStream);
250
251         // To do-- enable an IP-level trace that shows forwarding events only
252     }
253
254     // Give OLSR time to converge-- 30 seconds perhaps
255     Simulator::Schedule (Seconds (120.0), &GenerateTraffic,
256                          source, packetSize, numPackets, interPacketInterval);
257
258     // Output what we are doing
259     NS_LOG_UNCOND ("Testing from node " << sourceNode << " to " << sinkNode ...
            << " with grid distance " << distance);
260
261     Simulator::Stop (Seconds (123.0));
262
263     AnimationInterface anim("olsr-adhoc.xml");
264
265     Simulator::Run ();
266     Simulator::Destroy ();
267
268     return 0;
269 }
```