ETWS-1609 雷达系统

网络管理设计方案

部　　门：研发部

编　　制：罗敏

使用 LaTeX 撰写于 2018 年 5 月 18 日

摘　要

本文主要是关于雷达系统远程控制网络的体系结构设计，目的是实现全国各地雷达系统的远程监测与控制，主要是从两个方面来考虑如何设计实现，第一是从可操作性方面来考虑,因整个控制网连接的设备较多，特别是雷达设备甚至可能是位于郊区等人迹罕至的地方，将所有的接入设备都连接固定 IP 专网接口是不太现实的，而且费用过高，方案必须符合实际具有可操作性。第二个是从安全性的角度，整个系统分布与全国各地，且连接的雷达用户各不相同，需要考虑用户之间的隔离，以及防止非法的网络入侵破坏雷达系统工作。要实现这两个目的，通过可以采用 VPN 技术,将分布于各地的雷达设备、控制主机和后端管理服务器配置成为一个虚拟专网，这样的设计只需要位于后端的 VPN 服务器一个固定 IP，其它设备只需要连接普通的互联网服务便可以实现目的，费用低廉具有可操作性,且 VPN 的数据传输采用了多重加密技术可以有效防止数据泄露。

关键字：　远程监控　VPN　虚拟专网　安全性

目录

第一章 设计方案

1.1 系统结构

要对整个雷达系统进行网络设计，则需要先弄楚整个雷达系统的结构组成，下面主要对雷达系统的特点进行了一些总结：

- 系统庞大设备多，包括雷达、控制服务器、监控摄像头、UPS 设备等等
- 设备分散位于全国各地，后期可能还会不停的在各个地方扩展
- 需要时时监控设备工作情况，随时获取雷达数据

整个系统的结构如图 1.1 所示：

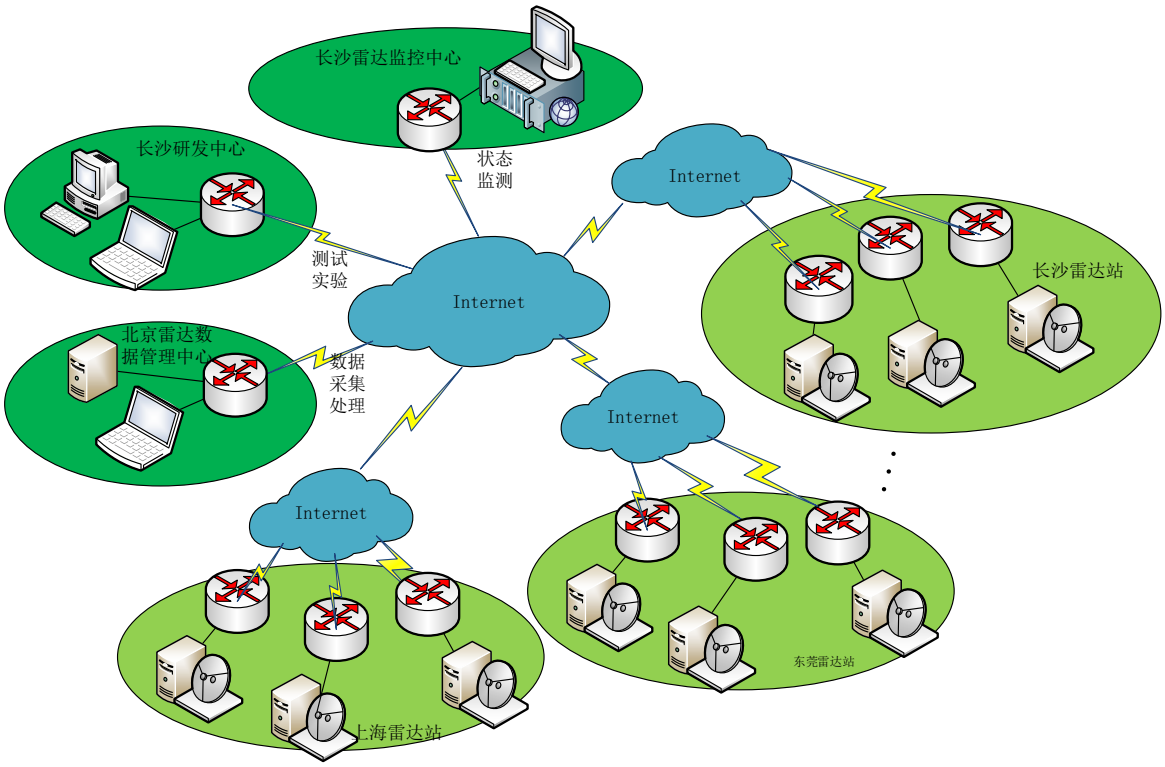

图 1.1: 雷达系统结构图

在图 1.1 可以看到雷达系统组成结构复杂，每个站点有多台雷达，不同站点的雷达也是各自分散的，根据目前的发展情况，后续还会增加更多的雷达站点，除了雷达站点之外雷达系统的监控管理中心在长沙，长沙的研发中心也会有访问雷达站点进行实验测试的需求，北京雷达数据管理中心则是主要需要从各个雷达站点读取数据进行雷达数据的处理。

1.2 网络设计

上一节简单的介绍了整个雷达系统的结构，本节根据雷达系统的结构, 进行网络管理方案设计。针对目前的情况和现有的技术，本文决定采用 VPN 技术组建私有专网，具体组网方案如图 1.2 所示：

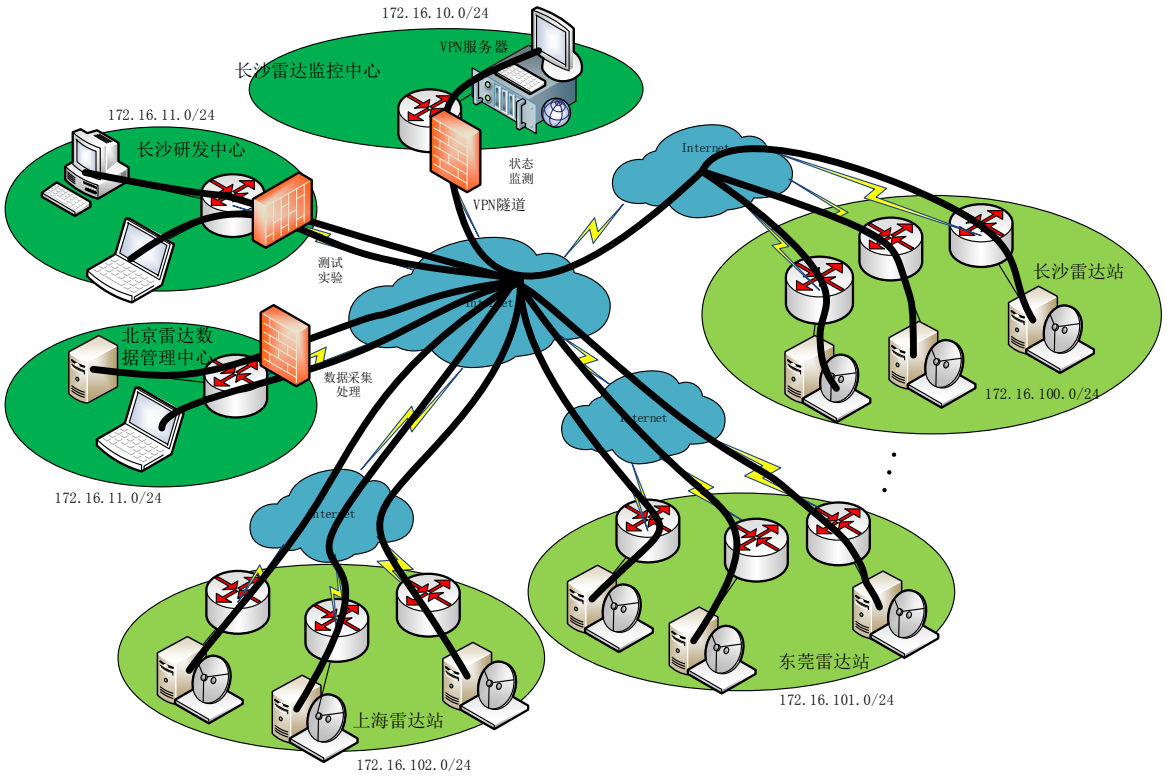

图 1.2: 雷达系统网络设计

在长沙雷达管理中心设置一台 VPN 服务器，来提供 VPN 服务管理，其它的网络节点，则通过 VPN 客户端远程加入 VPN 网络，因此长沙雷达管理中心需要一个专网 IP，以便其它节点可以方便访问。为了系统安全还需要进行网络隔离，各个雷达站之间不能相互访问，同

时需要在各个管理中心添加防火墙，防止雷达站访问到网络中心。

目前考虑没有能够满足要求的集成 VPN 路由器，只能采用在雷达设备的内部服务器上配置 VPN 客户端，后期可以考虑我司自主设计一个多功能网关设备作为雷达站点的控制网关，届时可以将 VPN 客户端配置在多功能集成网关上。

1.3 网络划分

在图 1.2 中标注的网络地址是指

第二章　网络安全

第三章　工具软件

第四章　配置步骤

4.1　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。

4.2　实验工具

同上一实验。

4.3　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。

4.4　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。

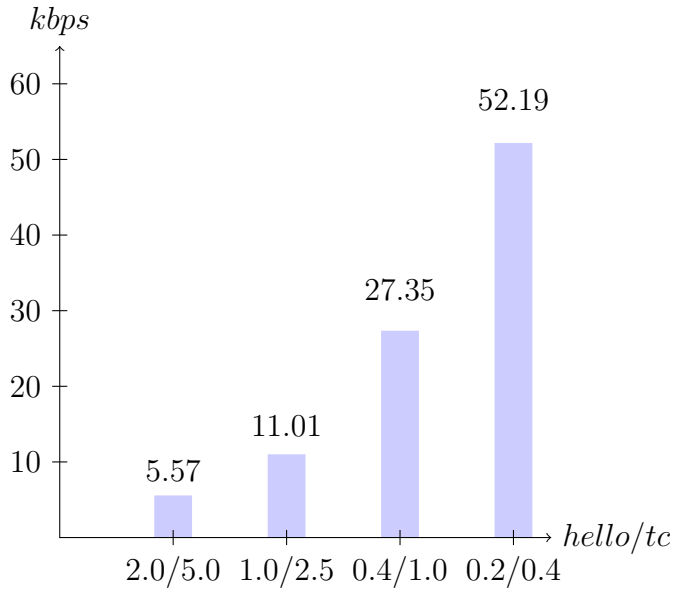

图 4.1: 报文数据流量

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。

4.5 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。

第五章　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 }
```