Networking with Linux Lab
Module 2 & 3: Client Server Network topology using NS-3 and Animating the Network

Assignment 6: Write an ns3 program to simulate a mesh topology and analyze the .pcap file using Wireshark.

Aim: To simulate a mesh network topology using ns3 and analyze network traffic using Wireshark for performance evaluation and troubleshooting

Theory: Mesh Topology

Mesh topology is a type of network topology where each device in the network is connected directly to every other device, forming a mesh-like structure. In a full mesh topology, every node has a direct link to every other node, while in a partial mesh, only some nodes have direct links to every other node.

Mesh topologies are highly reliable and fault-tolerant, as there are multiple paths for data to travel, reducing the risk of network failure. However, they can be complex to set up and require more cabling and configuration compared to other topologies like star or bus.

Code:

> mesh.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2008,2009 IITP RAS
 *
 * 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
 *
 * Author: Kirill Andreev <andreev@iitp.ru>
```

```
 *
 *
 * By default this script creates m_xSize * m_ySize square grid topology with
 * IEEE802.11s stack installed at each node with peering management
 * and HWMP protocol.
 * The side of the square cell is defined by m_step parameter.
 * When topology is created, UDP ping is installed to opposite corners
 * by diagonals. packet size of the UDP ping and interval between two
 * successive packets is configurable.
 *
 *  m_xSize * step
 *  |<--------->|
 *   step
 *  |<--->|
 *  * --- * --- * <---Ping sink  _
 *  | \   |   / |                ^
 *  |   \ | /   |                |
 *  * --- * --- * m_ySize * step |
 *  |   / | \   |                |
 *  | /   |   \ |                |
 *  * --- * --- *                _
 *  ^ Ping source
 *
 *  See also MeshTest::Configure to read more about configurable
 *  parameters.
 */

#include <iostream>
#include <sstream>
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/mesh-module.h"
#include "ns3/mobility-module.h"
#include "ns3/mesh-helper.h"
#include "ns3/yans-wifi-helper.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("TestMeshScript");

/**
 * \ingroup mesh
 * \brief MeshTest class
```

```
 */
class MeshTest
{
public:
  /// Init test
  MeshTest ();
  /**
   * Configure test from command line arguments
   *
   * \param argc command line argument count
   * \param argv command line arguments
   */
  void Configure (int argc, char ** argv);
  /**
   * Run test
   * \returns the test status
   */
  int Run ();
private:
  int       m_xSize; ///< X size
  int       m_ySize; ///< Y size
  double          m_step; ///< step
  double          m_randomStart; ///< random start
  double          m_totalTime; ///< total time
  double          m_packetInterval; ///< packet interval
  uint16_t  m_packetSize; ///< packet size
  uint32_t  m_nIfaces; ///< number interfaces
  bool   m_chan; ///< channel
  bool   m_pcap; ///< PCAP
  bool   m_ascii; ///< ASCII
  std::string m_stack; ///< stack
  std::string m_root; ///< root
  /// List of network nodes
  NodeContainer nodes;
  /// List of all mesh point devices
  NetDeviceContainer meshDevices;
  /// Addresses of interfaces:
  Ipv4InterfaceContainer interfaces;
  /// MeshHelper. Report is not static methods
  MeshHelper mesh;
private:
  /// Create nodes and setup their mobility
  void CreateNodes ();
  /// Install internet m_stack on nodes
  void InstallInternetStack ();
  /// Install applications
```

```
  void InstallApplication ();
  /// Print mesh devices diagnostics
  void Report ();
};
MeshTest::MeshTest () :
  m_xSize (3),
  m_ySize (3),
  m_step (100.0),
  m_randomStart (0.1),
  m_totalTime (100.0),
  m_packetInterval (0.1),
  m_packetSize (1024),
  m_nIfaces (1),
  m_chan (true),
  m_pcap (false),
  m_ascii (false),
  m_stack ("ns3::Dot11sStack"),
  m_root ("ff:ff:ff:ff:ff:ff")
{
}
void
MeshTest::Configure (int argc, char *argv[])
{
  CommandLine cmd (__FILE__);
  cmd.AddValue ("x-size", "Number of nodes in a row grid", m_xSize);
  cmd.AddValue ("y-size", "Number of rows in a grid", m_ySize);
  cmd.AddValue ("step",   "Size of edge in our grid (meters)", m_step);
  // Avoid starting all mesh nodes at the same time (beacons may collide)
  cmd.AddValue ("start",  "Maximum random start delay for beacon jitter (sec)", m_randomStart);
  cmd.AddValue ("time",  "Simulation time (sec)", m_totalTime);
  cmd.AddValue ("packet-interval",  "Interval between packets in UDP ping (sec)", m_packetInterval);
  cmd.AddValue ("packet-size",  "Size of packets in UDP ping (bytes)", m_packetSize);
  cmd.AddValue ("interfaces", "Number of radio interfaces used by each mesh point", m_nIfaces);
  cmd.AddValue ("channels",   "Use different frequency channels for different interfaces", m_chan);
  cmd.AddValue ("pcap",   "Enable PCAP traces on interfaces", m_pcap);
  cmd.AddValue ("ascii",   "Enable Ascii traces on interfaces", m_ascii);
  cmd.AddValue ("stack",  "Type of protocol stack. ns3::Dot11sStack by default", m_stack);
  cmd.AddValue ("root", "Mac address of root mesh point in HWMP", m_root);

  cmd.Parse (argc, argv);
  NS_LOG_DEBUG ("Grid:" << m_xSize << "*" << m_ySize);
  NS_LOG_DEBUG ("Simulation time: " << m_totalTime << " s");
```

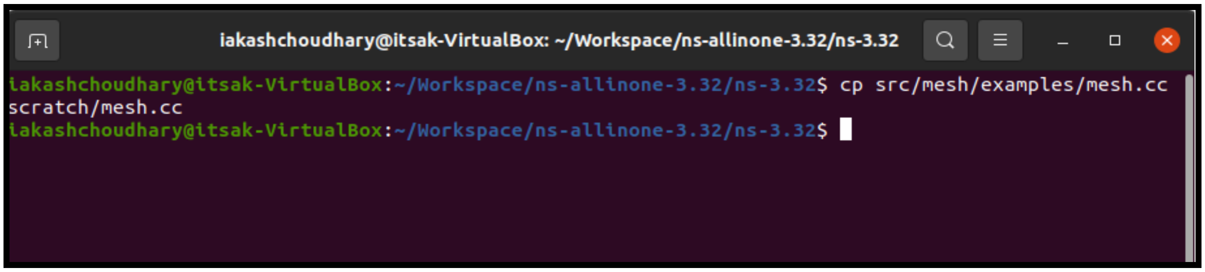
```
  if (m_ascii)
        {
        PacketMetadata::Enable ();
        }
}
void
MeshTest::CreateNodes ()
{
  /*
   * Create m_ySize*m_xSize stations to form a grid topology
   */
  nodes.Create (m_ySize*m_xSize);
  // Configure YansWifiChannel
  YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
  YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
  wifiPhy.SetChannel (wifiChannel.Create ());
  /*
   * Create mesh helper and set stack installer to it
   * Stack installer creates all needed protocols and install them to
   * mesh point device
   */
  mesh = MeshHelper::Default ();
  if (!Mac48Address (m_root.c_str ()).IsBroadcast ())
        {
        mesh.SetStackInstaller (m_stack, "Root", Mac48AddressValue (Mac48Address
(m_root.c_str ())));
        }
  else
        {
        //If root is not set, we do not use "Root" attribute, because it
        //is specified only for 11s
        mesh.SetStackInstaller (m_stack);
        }
  if (m_chan)
        {
        mesh.SetSpreadInterfaceChannels (MeshHelper::SPREAD_CHANNELS);
        }
  else
        {
        mesh.SetSpreadInterfaceChannels (MeshHelper::ZERO_CHANNEL);
        }
  mesh.SetMacType ("RandomStart", TimeValue (Seconds (m_randomStart)));
  // Set number of interfaces - default is single-interface mesh point
  mesh.SetNumberOfInterfaces (m_nIfaces);
  // Install protocols and return container if MeshPointDevices
  meshDevices = mesh.Install (wifiPhy, nodes);
```

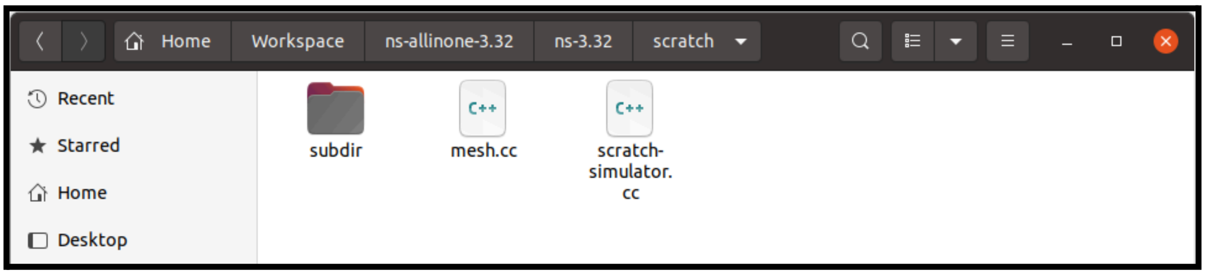
```
  // Setup mobility - static grid topology
  MobilityHelper mobility;
  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                 "MinX", DoubleValue (0.0),
                                 "MinY", DoubleValue (0.0),
                                 "DeltaX", DoubleValue (m_step),
                                 "DeltaY", DoubleValue (m_step),
                                 "GridWidth", UintegerValue (m_xSize),
                                 "LayoutType", StringValue ("RowFirst"));
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (nodes);
  if (m_pcap)
        wifiPhy.EnablePcapAll (std::string ("mp-"));
  if (m_ascii)
        {
        AsciiTraceHelper ascii;
        wifiPhy.EnableAsciiAll (ascii.CreateFileStream ("mesh.tr"));
        }
}
void
MeshTest::InstallInternetStack ()
{
  InternetStackHelper internetStack;
  internetStack.Install (nodes);
  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");
  interfaces = address.Assign (meshDevices);
}
void
MeshTest::InstallApplication ()
{
  UdpEchoServerHelper echoServer (9);
  ApplicationContainer serverApps = echoServer.Install (nodes.Get (0));
  serverApps.Start (Seconds (0.0));
  serverApps.Stop (Seconds (m_totalTime));
  UdpEchoClientHelper echoClient (interfaces.GetAddress (0), 9);
                                echoClient.SetAttribute ("MaxPackets", UintegerValue
((uint32_t)(m_totalTime*(1/m_packetInterval))));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (m_packetInterval)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (m_packetSize));
  ApplicationContainer clientApps = echoClient.Install (nodes.Get (m_xSize*m_ySize-1));
  clientApps.Start (Seconds (0.0));
  clientApps.Stop (Seconds (m_totalTime));
}
int
MeshTest::Run ()
```

```
{
  CreateNodes ();
  InstallInternetStack ();
  InstallApplication ();
  Simulator::Schedule (Seconds (m_totalTime), &MeshTest::Report, this);
  Simulator::Stop (Seconds (m_totalTime));
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
void
MeshTest::Report ()
{
  unsigned n (0);
   for (NetDeviceContainer::Iterator i = meshDevices.Begin (); i != meshDevices.End (); ++i,
++n)
        {
        std::ostringstream os;
        os << "mp-report-" << n << ".xml";
        std::cerr << "Printing mesh point device #" << n << " diagnostics to " << os.str () <<
"\n";
        std::ofstream of;
        of.open (os.str ().c_str ());
        if (!of.is_open ())
        {
        std::cerr << "Error: Can't open file " << os.str () << "\n";
        return;
        }
        mesh.Report (*i, of);
        of.close ();
        }
}
int
main (int argc, char *argv[])
{
  MeshTest t;
  t.Configure (argc, argv);
  return t.Run ();
}
```

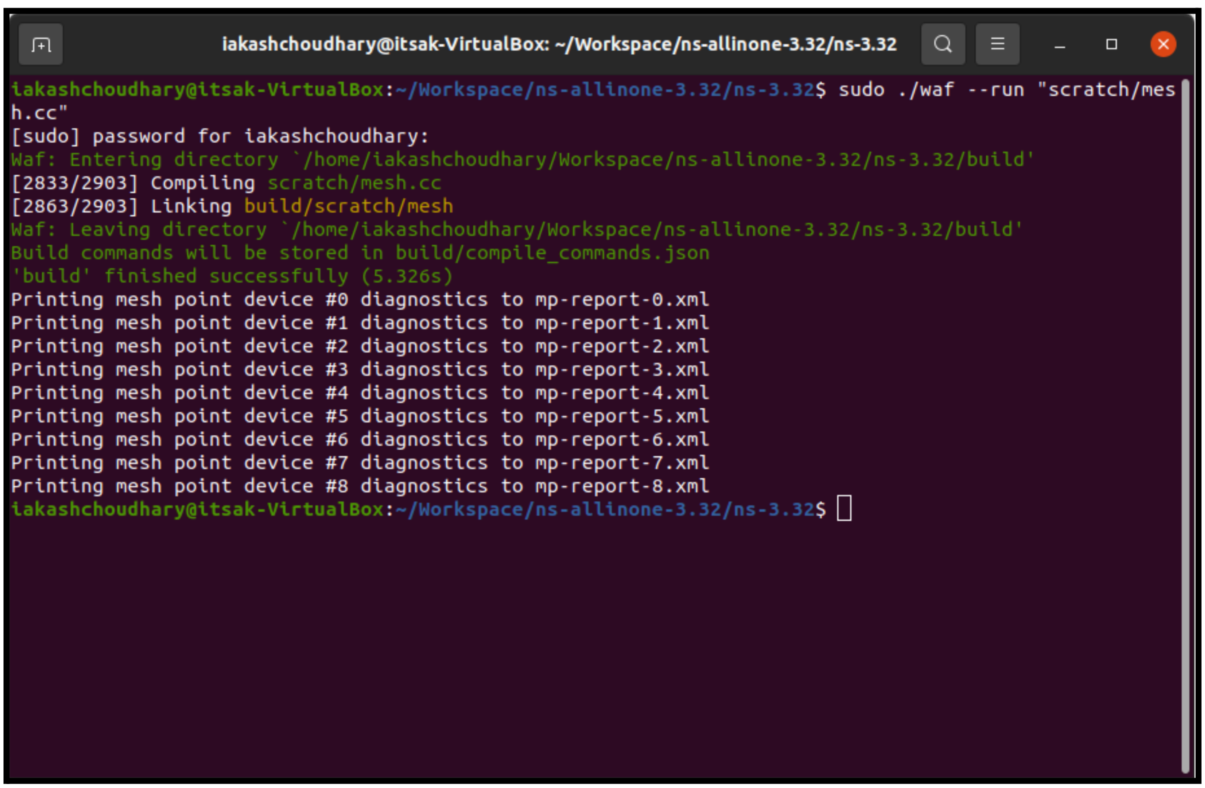
Command & Screenshot:

> $ cp src/mesh/examples/mesh.cc scratch/mesh.cc

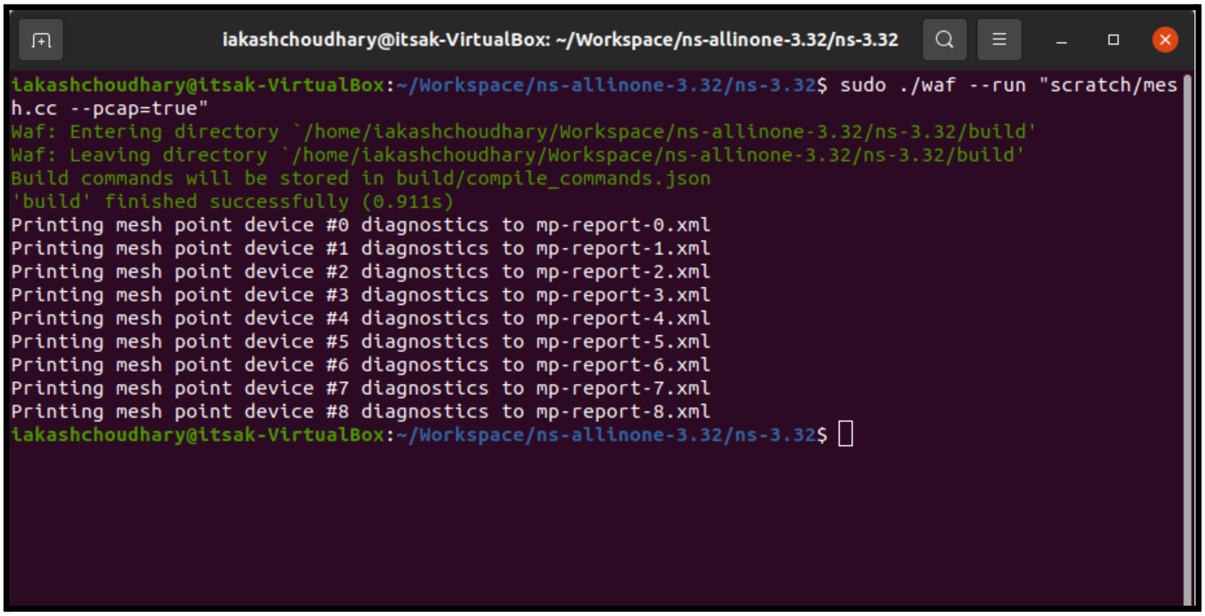




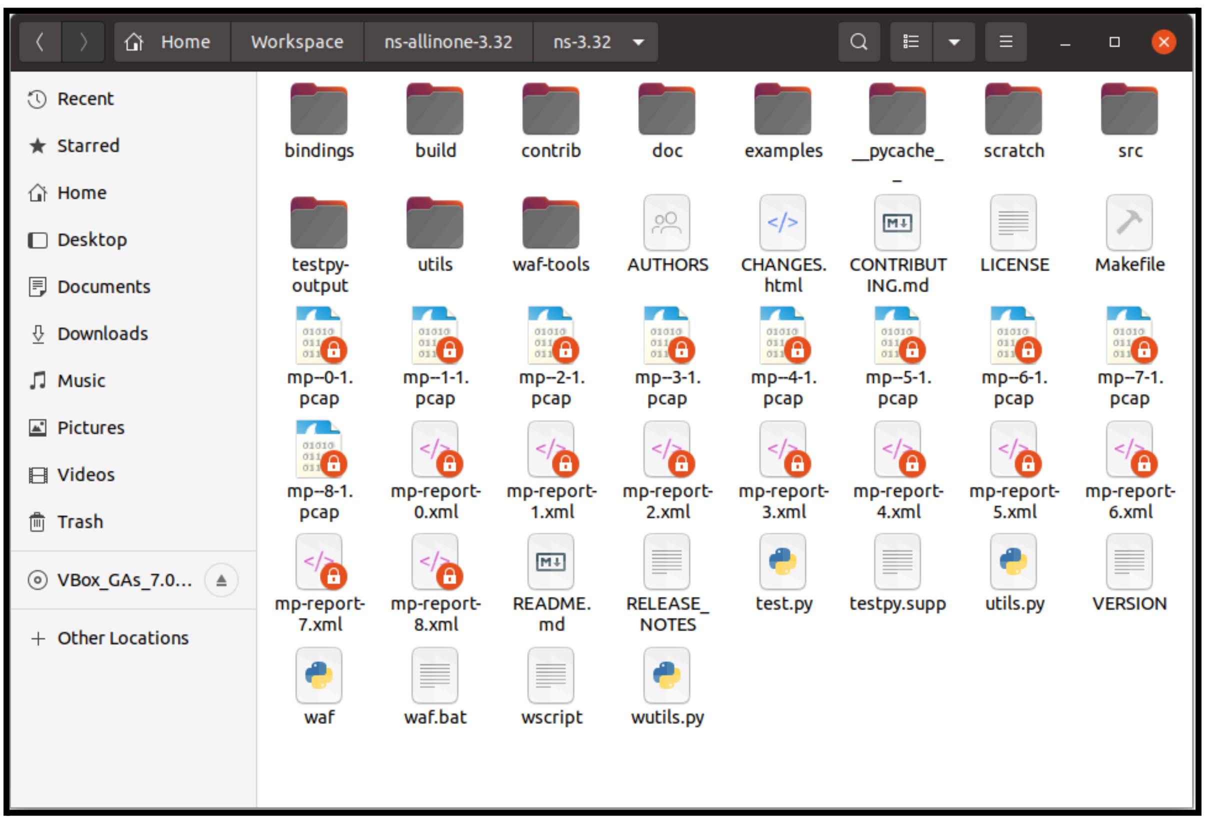
> $ sudo ./waf --run "scratch/mesh.cc"



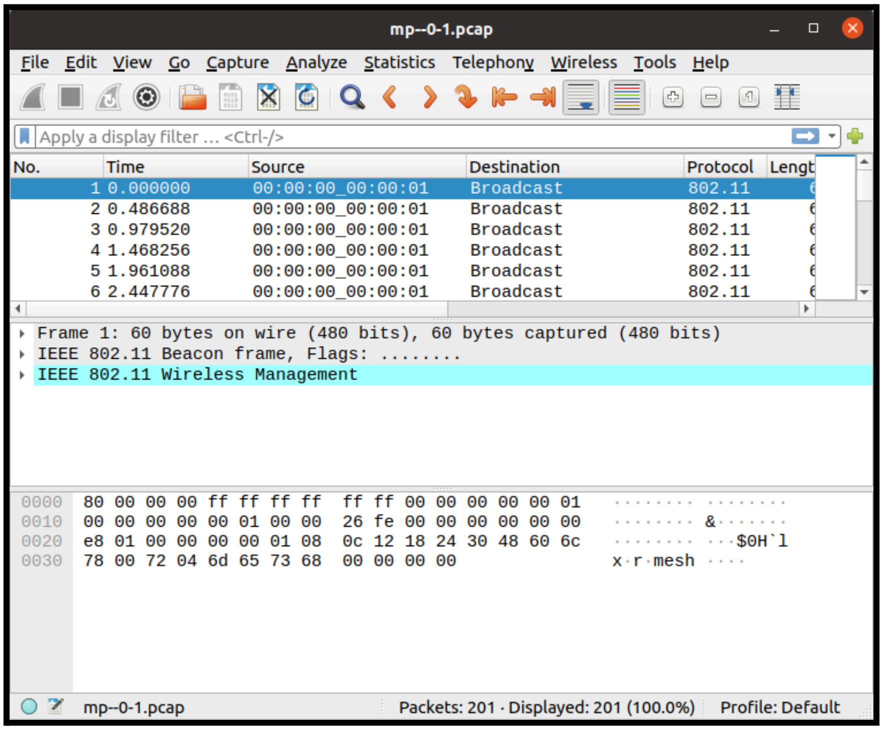
> **$ sudo ./waf --run "scratch/mesh.cc --pcap=true"**

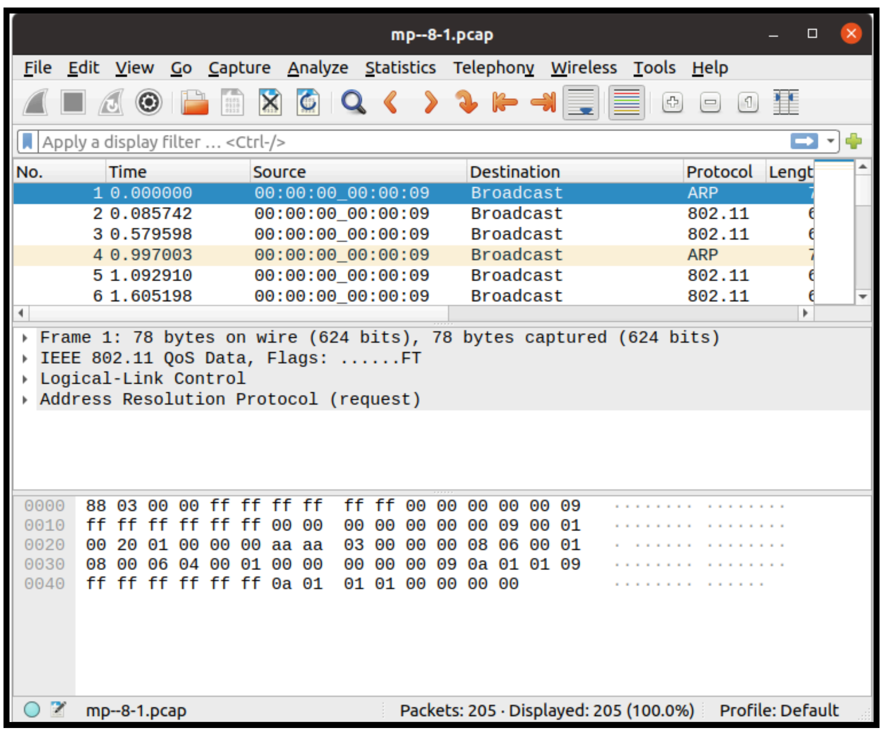


> After executing the above command, a **.pcap file is created** for Wireshark analysis.



> To open each .pcap file, simply **double-click** on it.





Conclusion: Analyzing the simulated mesh network using Wireshark reveals the efficiency and performance of communication among interconnected nodes.