NS LAB 28 AUGUST | 120CS0124 Maloth Aditya

Name: Maloth Aditya
Roll No: 120CS0124

Analyze CSMA-broadcast, CSMA-multicast, and CSMA-one-subnet.
Do the result analysis via, .xml, .tr, and .pcap files generated from the code run.
Display the result in NetAnim, Tracemetrics, and Wireshark.

Broadcast CODE:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
* 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
 */

//
// Example of the sending of a datagram to a broadcast address
//
// Network topology
//    ==============
//      |        |
//      n0    n1   n2
//      |     |
//    ==========
//
//   n0 originates UDP broadcast to 255.255.255.255/discard port, which
//   is replicated and received on both n1 and n2

#include <iostream>
#include <fstream>
#include <string>
#include <cassert>

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/netanim-module.h"
```

NS LAB 28 AUGUST | 120CS0124 Maloth Aditya

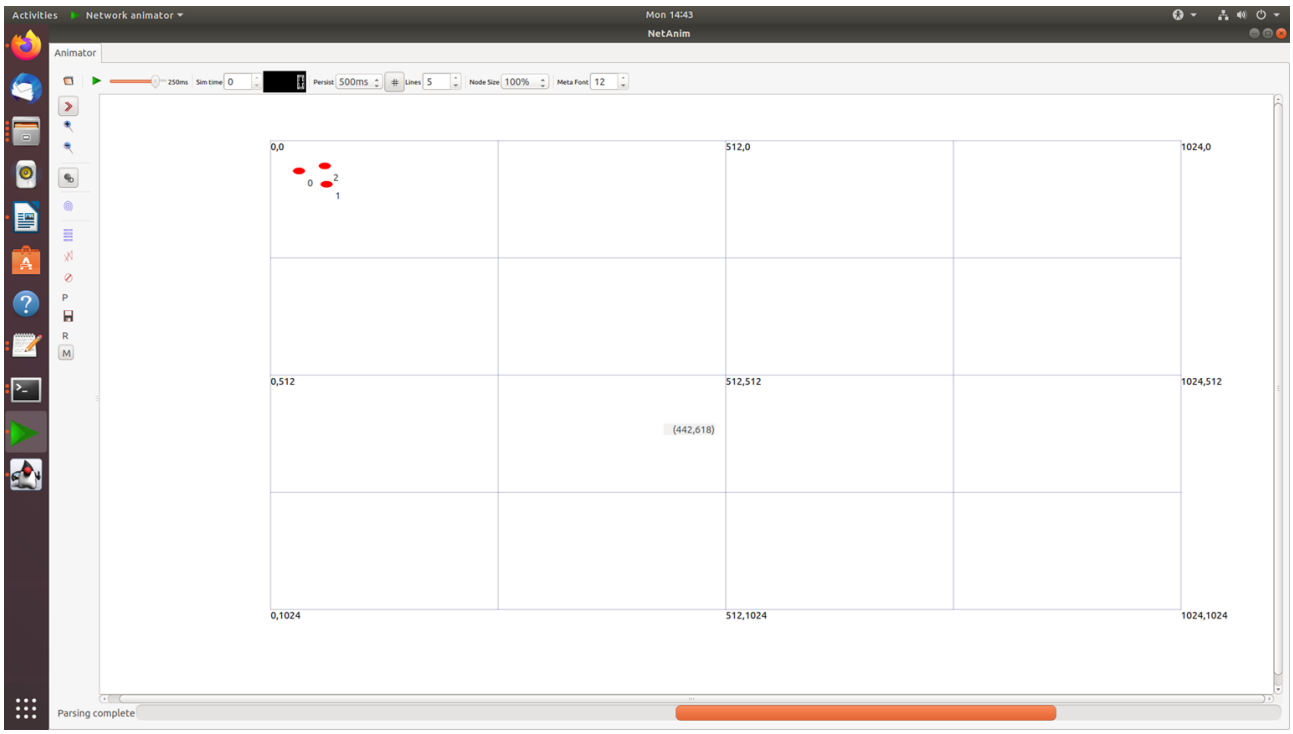
```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("CsmaBroadcastExample");

int main (int argc, char *argv[]){
 // Users may find it convenient to turn on explicit debugging
 // for selected modules; the below lines suggest how to do this
 #if 0
 LogComponentEnable ("CsmaBroadcastExample", LOG_LEVEL_INFO);
 #endif
 LogComponentEnable ("CsmaBroadcastExample", LOG_PREFIX_TIME);

 // Allow the user to override any of the defaults and the above
 // Bind()s at run-time, via command-line arguments
 CommandLine cmd;
 cmd.Parse (argc, argv);

 NS_LOG_INFO ("Create nodes.");
 NodeContainer c;
 c.Create (3);
 NodeContainer c0 = NodeContainer (c.Get (0), c.Get (1));
 NodeContainer c1 = NodeContainer (c.Get (0), c.Get (2));

 NS_LOG_INFO ("Build Topology.");
 CsmaHelper csma;
 csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate (5000000)));
 csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));

 NetDeviceContainer n0 = csma.Install (c0);
 NetDeviceContainer n1 = csma.Install (c1);

 InternetStackHelper internet;
 internet.Install (c);

 NS_LOG_INFO ("Assign IP Addresses.");
 Ipv4AddressHelper ipv4;
 ipv4.SetBase ("10.1.0.0", "255.255.255.0");
 ipv4.Assign (n0);
 ipv4.SetBase ("192.168.1.0", "255.255.255.0");
 ipv4.Assign (n1);


 // RFC 863 discard port ("9") indicates packet should be thrown away
 // by the system.  We allow this silent discard to be overridden
 // by the PacketSink application.
 uint16_t port = 9;

 // Create the OnOff application to send UDP datagrams of size
```

```
  // 512 bytes (default) at a rate of 500 Kb/s (default) from n0
  NS_LOG_INFO ("Create Applications.");
  OnOffHelper onoff ("ns3::UdpSocketFactory", Address (InetSocketAddress (Ipv4Address
("255.255.255.255"), port)));
  onoff.SetConstantRate (DataRate ("500kb/s"));

  ApplicationContainer app = onoff.Install (c0.Get (0));
  // Start the application
  app.Start (Seconds (1.0));
  app.Stop (Seconds (10.0));

  // Create an optional packet sink to receive these packets
  PacketSinkHelper sink ("ns3::UdpSocketFactory",Address (InetSocketAddress
(Ipv4Address::GetAny (), port)));
  app = sink.Install (c0.Get (1));
  app.Add (sink.Install (c1.Get (1)));
  app.Start (Seconds (1.0));
  app.Stop (Seconds (10.0));

  //Configure node positions
  AnimationInterface anim("csma-broadcasr.xml");
  anim.SetConstantPosition(c.Get(0), 10.0,10.0);
  anim.SetConstantPosition(c.Get(1), 25.0,25.0);
  anim.SetConstantPosition(c.Get(2), 50.0,50.0);

  // Configure ascii tracing of all enqueue, dequeue, and NetDevice receive
  // events on all devices.  Trace output will be sent to the file
  // "csma-one-subnet.tr"
  AsciiTraceHelper ascii;
  csma.EnableAsciiAll (ascii.CreateFileStream ("csma-broadcast.tr"));

  // Also configure some tcpdump traces; each interface will be traced
  // The output files will be named
  // csma-broadcast-<nodeId>-<interfaceId>.pcap
  // and can be read by the "tcpdump -tt -r" command
  csma.EnablePcapAll ("csma-broadcast", false);

  NS_LOG_INFO ("Run Simulation.");
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
 }
```

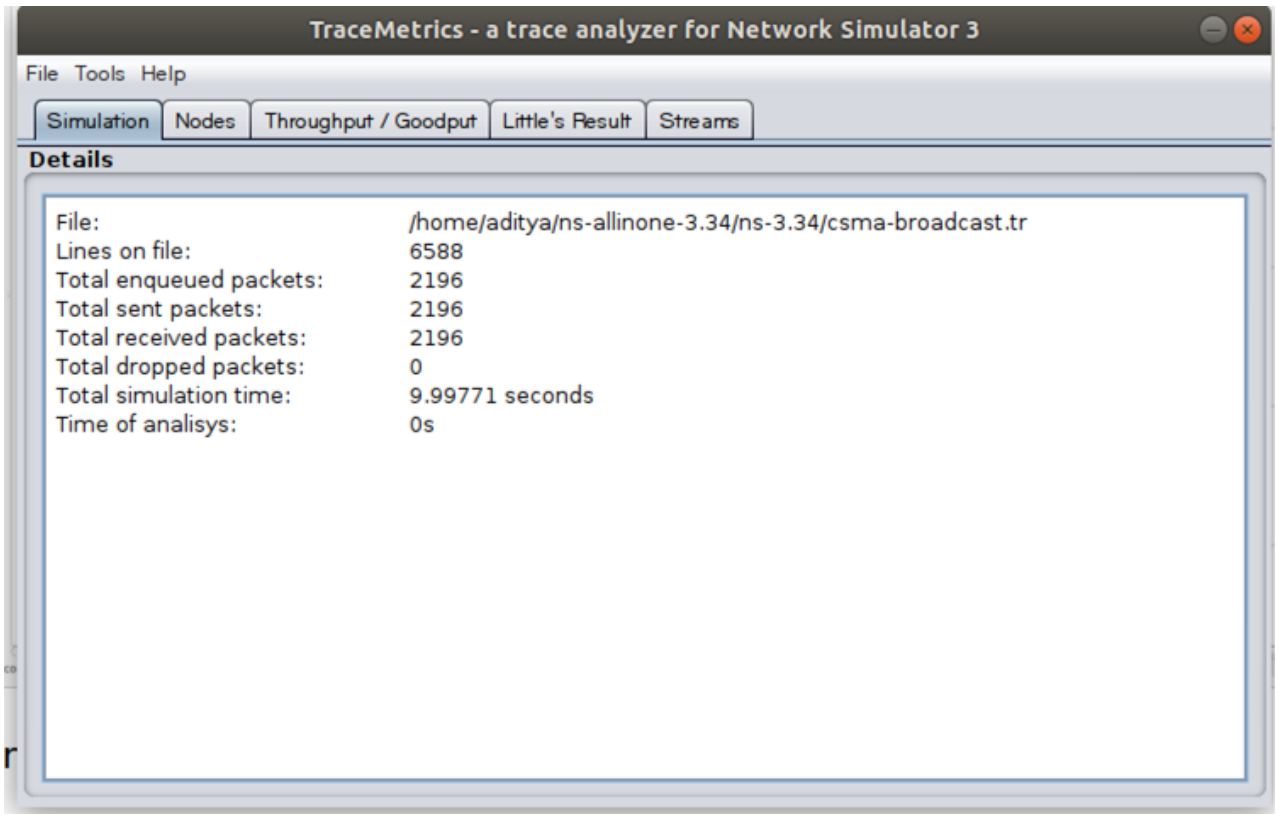
OUTPUT:

NetAnim:

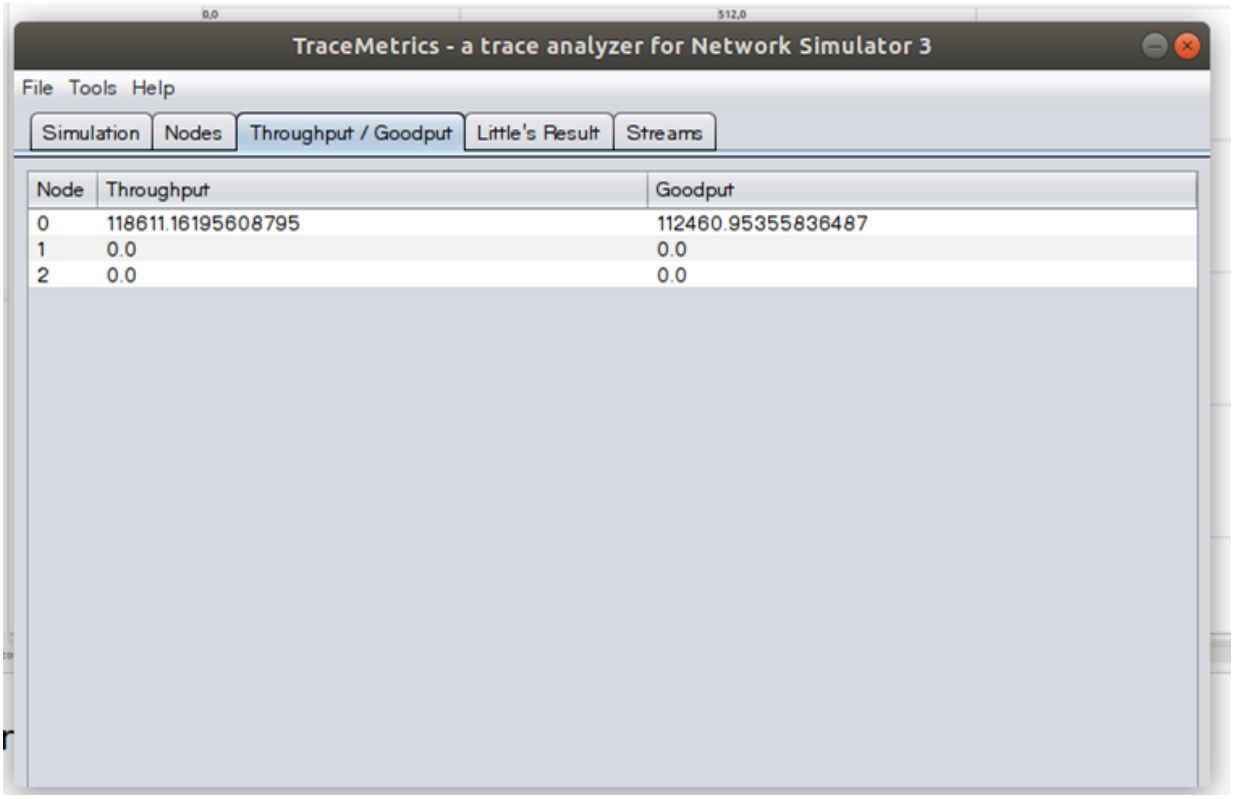
NS LAB 28 AUGUST | 120CS0124 Maloth Aditya



Tracemetrics:

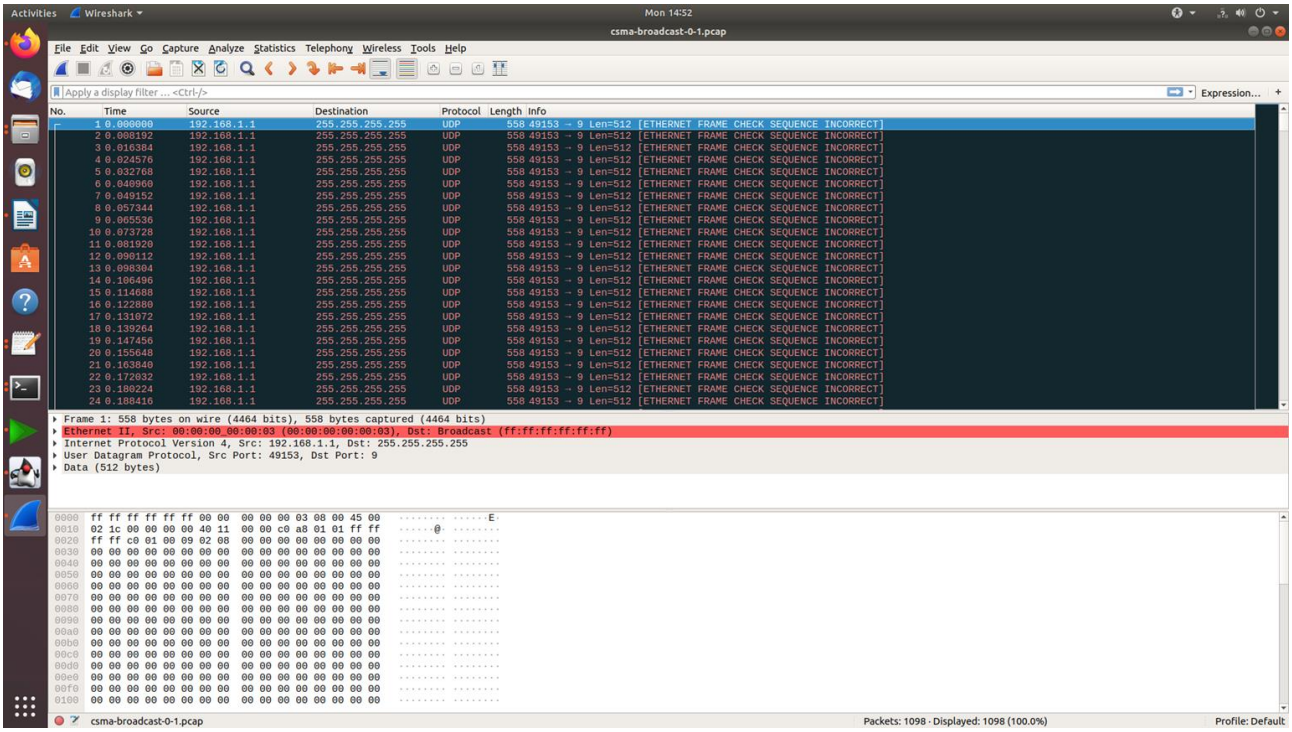


NS LAB 28 AUGUST | 120CS0124 Maloth Aditya

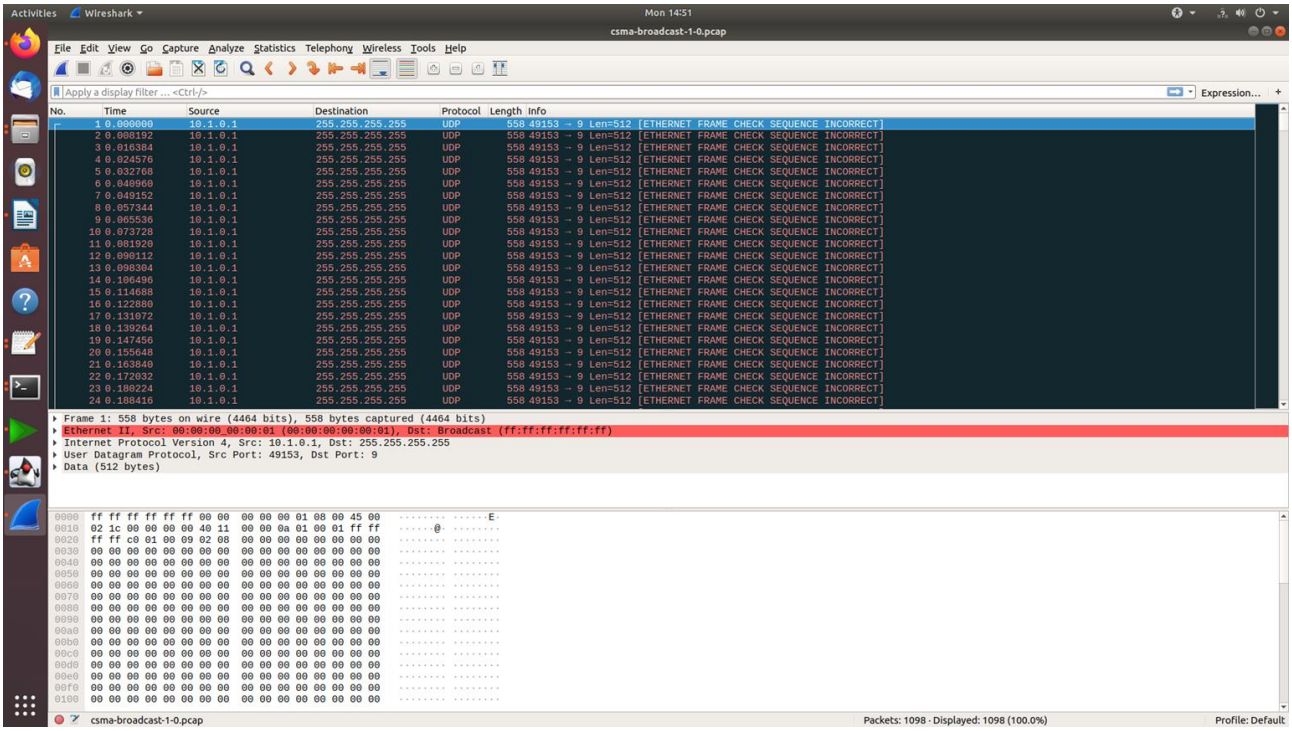


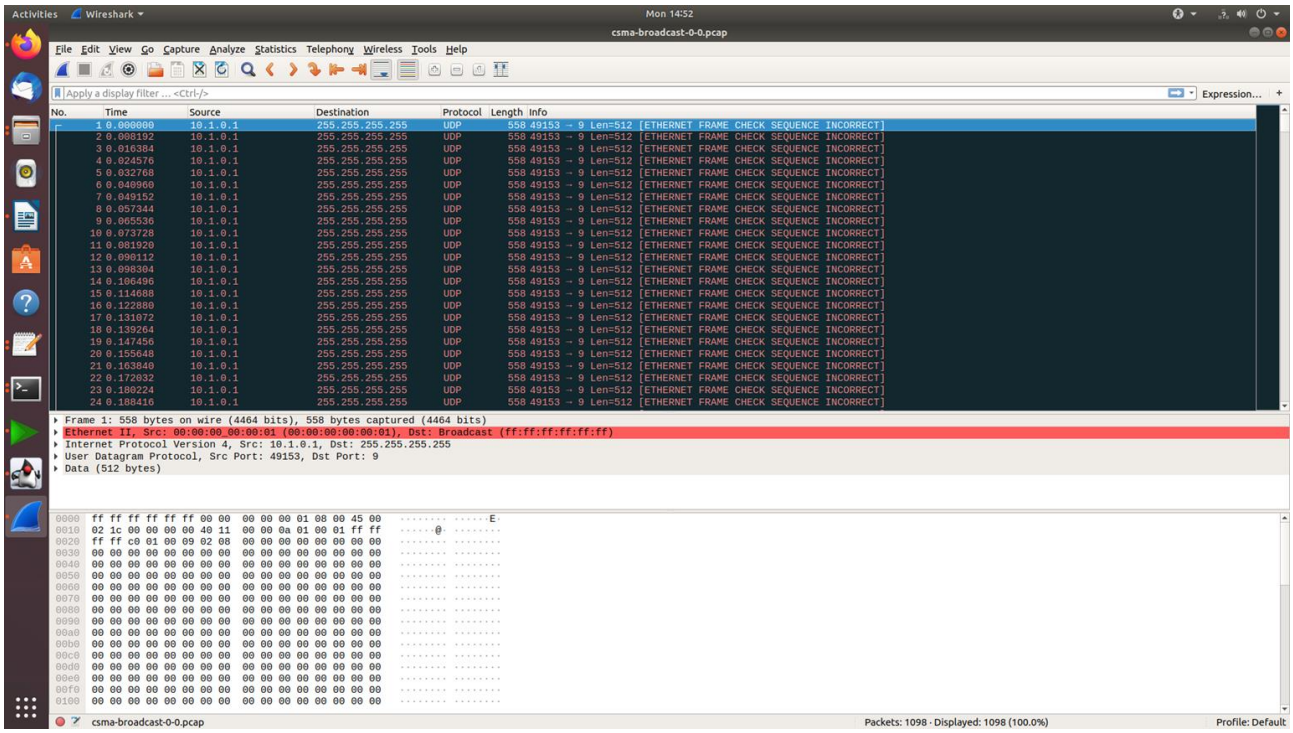
Node	Throughput	Goodput
0	118611.16195608795	112460.95355836487
1	0.0	0.0
2	0.0	0.0

Wireshark:

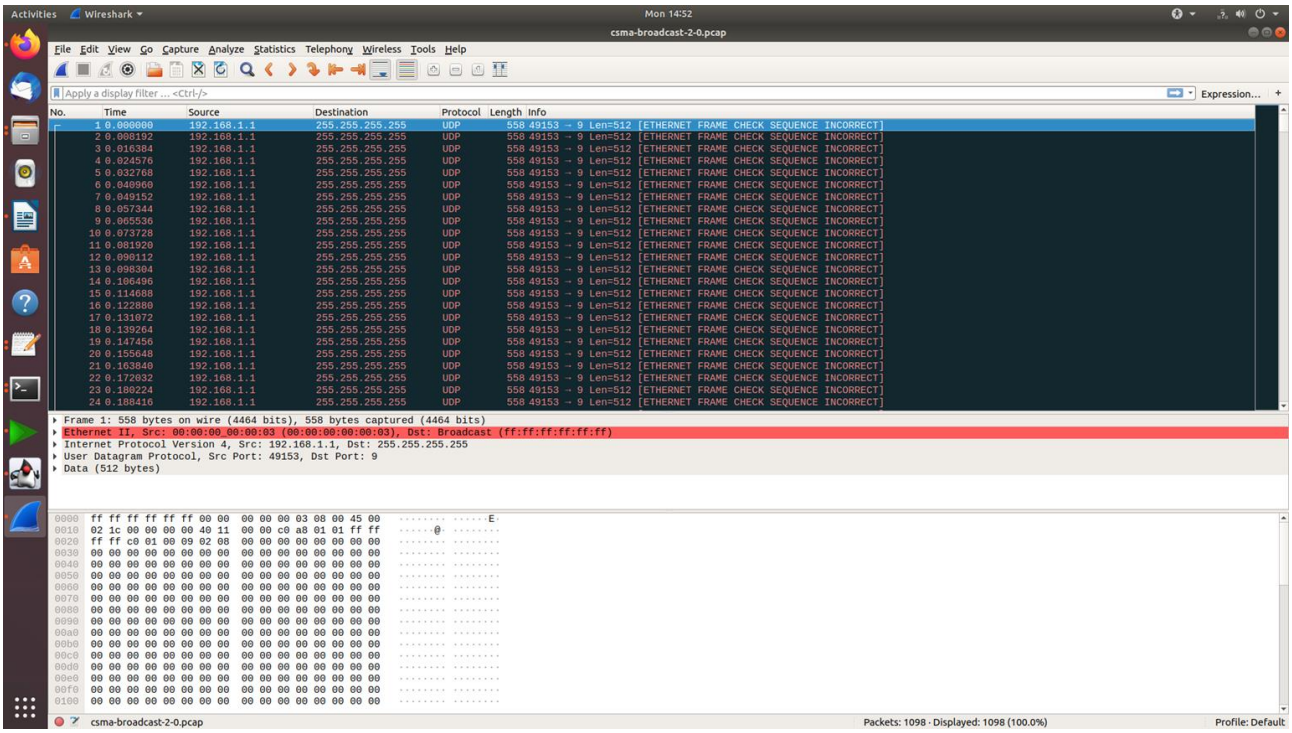


NS LAB 28 AUGUST | 120CS0124 Maloth Aditya





NS LAB 28 AUGUST | 120CS0124 Maloth Aditya



Multicast CODE:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * 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
 */

// Network topology
//
//             Lan1
//           ===========
//           |   |   |
//     n0  n1  n2  n3  n4
//     |   |   |
//     ===========
//        Lan0
//
// - Multicast source is at node n0;
```

```
// - Multicast forwarded by node n2 onto LAN1;
// - Nodes n0, n1, n2, n3, and n4 receive the multicast frame.
// - Node n4 listens for the data

#include <iostream>
#include <fstream>

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/netanim-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("CsmaMulticastExample");

int
main (int argc, char *argv[])
{
  //
  // Users may find it convenient to turn on explicit debugging
  // for selected modules; the below lines suggest how to do this
  //
  // LogComponentEnable ("CsmaMulticastExample", LOG_LEVEL_INFO);

  //
  // Set up default values for the simulation.
  //
  // Select DIX/Ethernet II-style encapsulation (no LLC/Snap header)
  Config::SetDefault ("ns3::CsmaNetDevice::EncapsulationMode", StringValue ("Dix"));

  // Allow the user to override any of the defaults at
  // run-time, via command-line arguments
  CommandLine cmd;
  cmd.Parse (argc, argv);

  NS_LOG_INFO ("Create nodes.");
  NodeContainer c;
  c.Create (5);
  // We will later want two subcontainers of these nodes, for the two LANs
  NodeContainer c0 = NodeContainer (c.Get (0), c.Get (1), c.Get (2));
  NodeContainer c1 = NodeContainer (c.Get (2), c.Get (3), c.Get (4));

  NS_LOG_INFO ("Build Topology.");
  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate (5000000)));
  csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
```

NS LAB 28 AUGUST | 120CS0124 Maloth Aditya

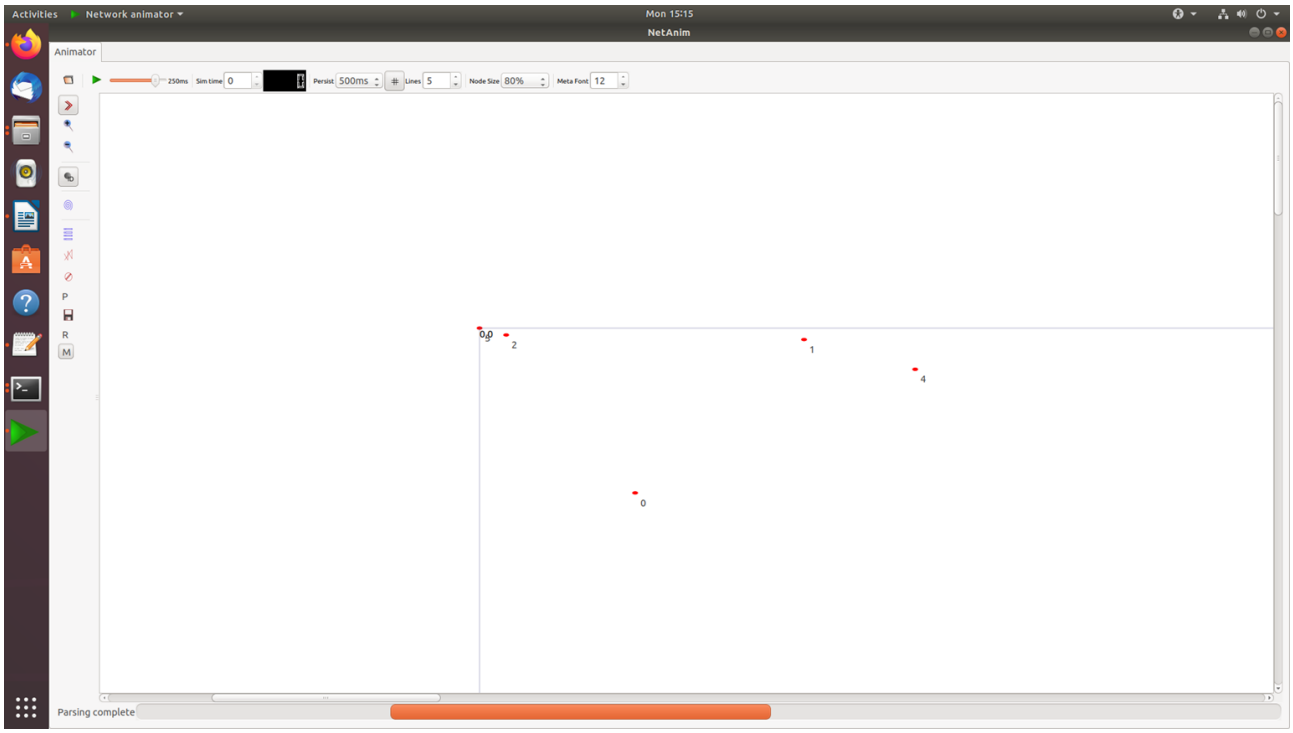
```
// We will use these NetDevice containers later, for IP addressing
NetDeviceContainer nd0 = csma.Install (c0);  // First LAN
NetDeviceContainer nd1 = csma.Install (c1);  // Second LAN

NS_LOG_INFO ("Add IP Stack.");
InternetStackHelper internet;
internet.Install (c);

NS_LOG_INFO ("Assign IP Addresses.");
Ipv4AddressHelper ipv4Addr;
ipv4Addr.SetBase ("10.1.1.0", "255.255.255.0");
ipv4Addr.Assign (nd0);
ipv4Addr.SetBase ("10.1.2.0", "255.255.255.0");
ipv4Addr.Assign (nd1);

NS_LOG_INFO ("Configure multicasting.");
//
// Now we can configure multicasting.  As described above, the multicast
// source is at node zero, which we assigned the IP address of 10.1.1.1
// earlier.  We need to define a multicast group to send packets to.  This
// can be any multicast address from 224.0.0.0 through 239.255.255.255
// (avoiding the reserved routing protocol addresses).
//

Ipv4Address multicastSource ("10.1.1.1");
Ipv4Address multicastGroup ("225.1.2.4");

// Now, we will set up multicast routing.  We need to do three things:
// 1) Configure a (static) multicast route on node n2
// 2) Set up a default multicast route on the sender n0
// 3) Have node n4 join the multicast group
// We have a helper that can help us with static multicast
Ipv4StaticRoutingHelper multicast;

// 1) Configure a (static) multicast route on node n2 (multicastRouter)
Ptr<Node> multicastRouter = c.Get (2);  // The node in question
Ptr<NetDevice> inputIf = nd0.Get (2);  // The input NetDevice
NetDeviceContainer outputDevices;  // A container of output NetDevices
outputDevices.Add (nd1.Get (0));  // (we only need one NetDevice here)

multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);

// 2) Set up a default multicast route on the sender n0
Ptr<Node> sender = c.Get (0);
Ptr<NetDevice> senderIf = nd0.Get (0);
multicast.SetDefaultMulticastRoute (sender, senderIf);
```

```
//
// Create an OnOff application to send UDP datagrams from node zero to the
// multicast group (node four will be listening).
//
NS_LOG_INFO ("Create Applications.");

uint16_t multicastPort = 9;   // Discard port (RFC 863)

// Configure a multicast packet generator that generates a packet
// every few seconds
OnOffHelper onoff ("ns3::UdpSocketFactory",
                   Address (InetSocketAddress (multicastGroup, multicastPort)));
onoff.SetConstantRate (DataRate ("255b/s"));
onoff.SetAttribute ("PacketSize", UintegerValue (128));

ApplicationContainer srcC = onoff.Install (c0.Get (0));

//
// Tell the application when to start and stop.
//
srcC.Start (Seconds (1.));
srcC.Stop (Seconds (10.));

// Create an optional packet sink to receive these packets
PacketSinkHelper sink ("ns3::UdpSocketFactory",
                       InetSocketAddress (Ipv4Address::GetAny (), multicastPort));

ApplicationContainer sinkC = sink.Install (c1.Get (2)); // Node n4
// Start the sink
sinkC.Start (Seconds (1.0));
sinkC.Stop (Seconds (10.0));


AnimationInterface anim("csma-multicast.xml");
anim.SetConstantPosition(c.Get(0), 10.0,10.0);
anim.SetConstantPosition(c.Get(1), 20.0,20.0);
anim.SetConstantPosition(c.Get(2), 15.0,15.0);
anim.SetConstantPosition(c.Get(3), 10.0,20.0);
anim.SetConstantPosition(c.Get(4), 20.0,10.0);

NS_LOG_INFO ("Configure Tracing.");
//
// Configure tracing of all enqueue, dequeue, and NetDevice receive events.
// Ascii trace output will be sent to the file "csma-multicast.tr"
//
AsciiTraceHelper ascii;
csma.EnableAsciiAll (ascii.CreateFileStream ("csma-multicast.tr"));

// Also configure some tcpdump traces; each interface will be traced.
```

```
  // The output files will be named:
  //     csma-multicast-<nodeId>-<interfaceId>.pcap
  // and can be read by the "tcpdump -r" command (use "-tt" option to
  // display timestamps correctly)
  csma.EnablePcapAll ("csma-multicast", false);

  //
  // Now, do the actual simulation.
  //
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
}
```

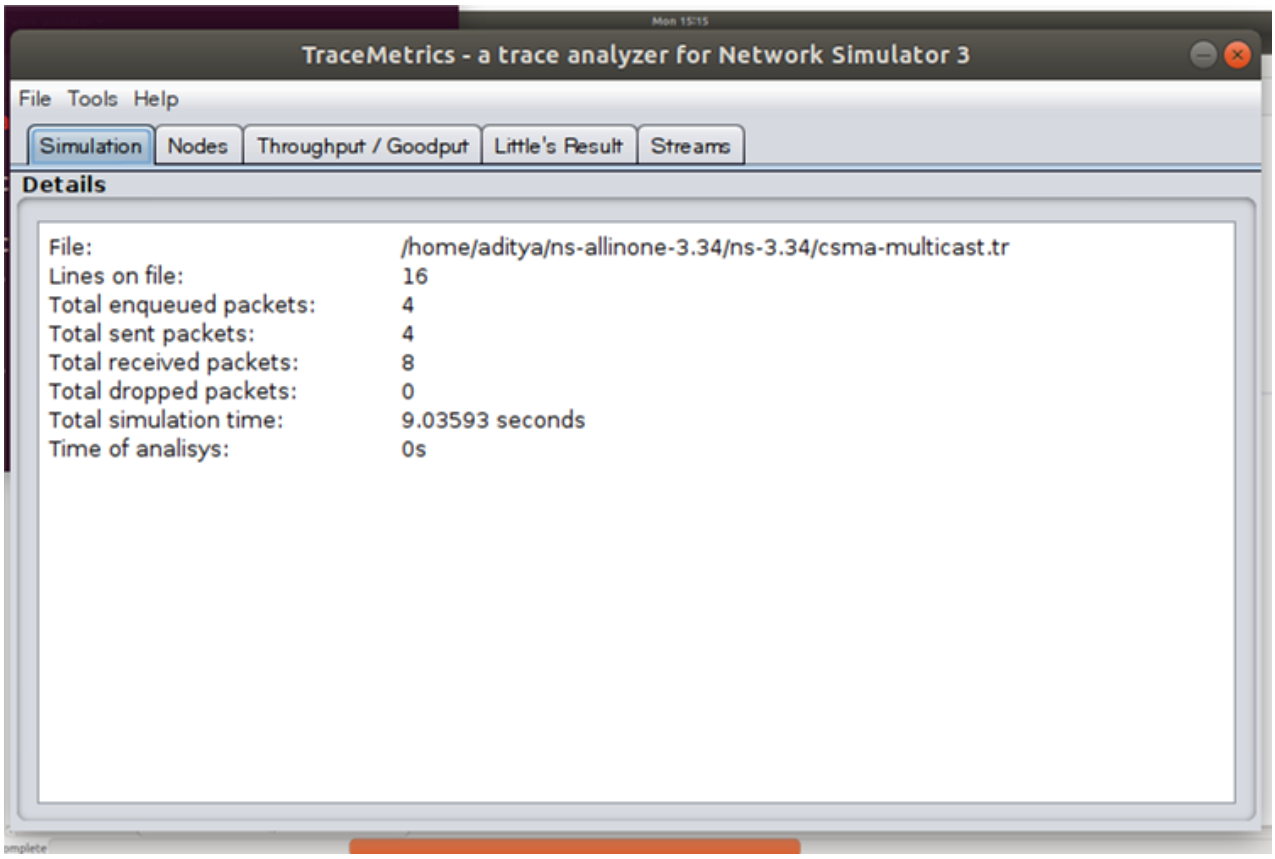
OUTPUT:

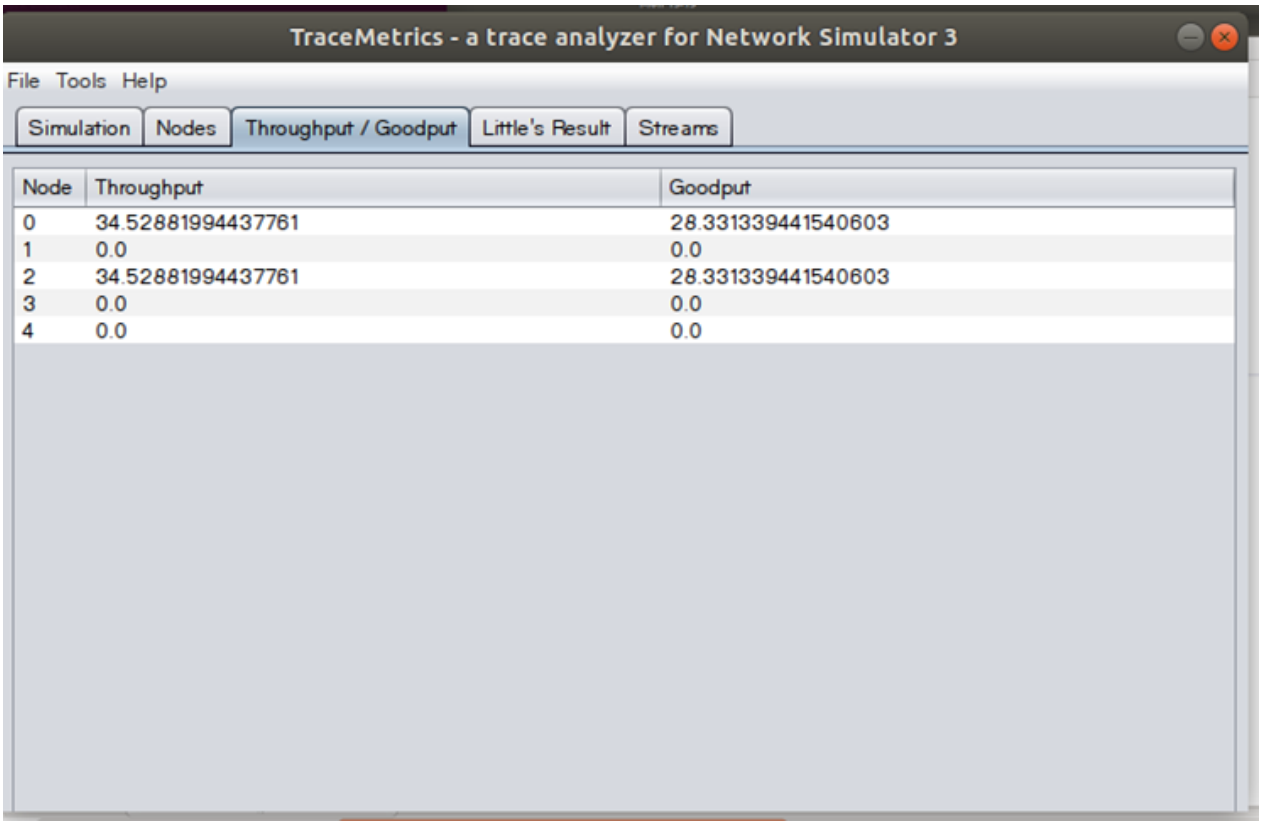
NetAnim:



Tracemetrics:

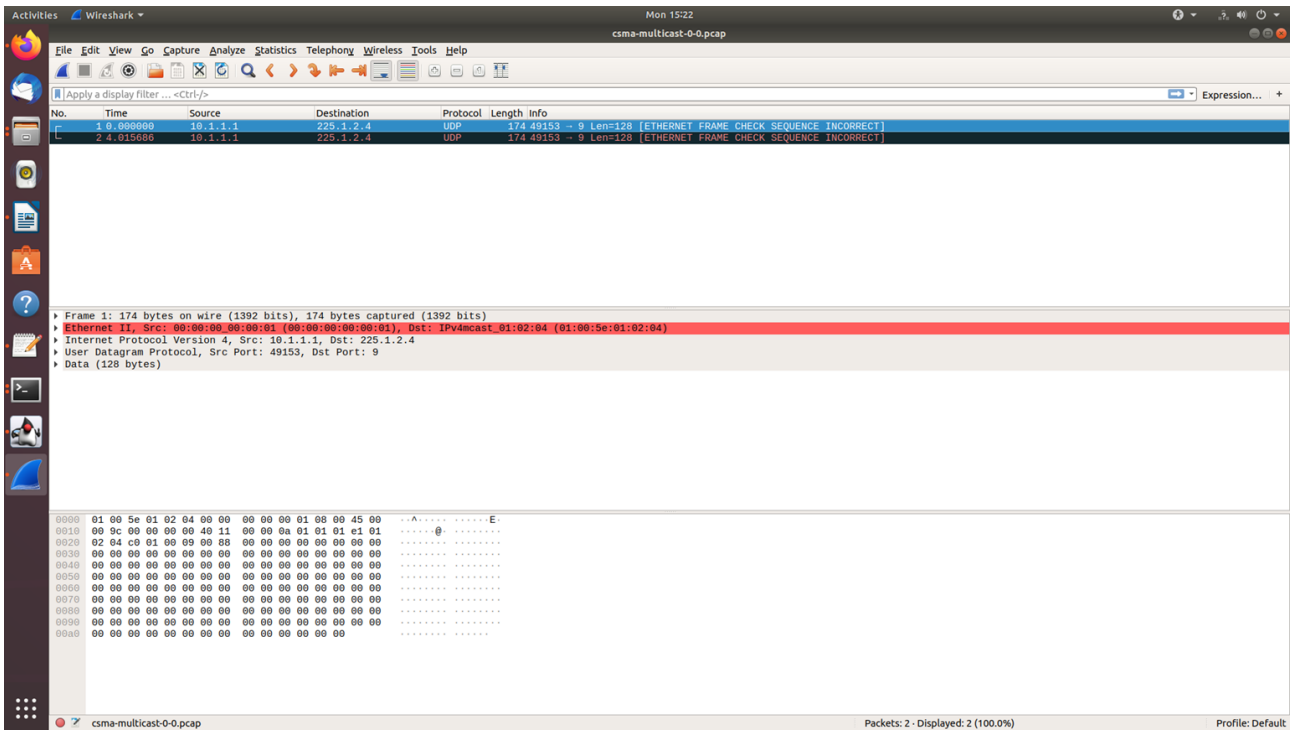
NS LAB 28 AUGUST | 120CS0124 Maloth Aditya

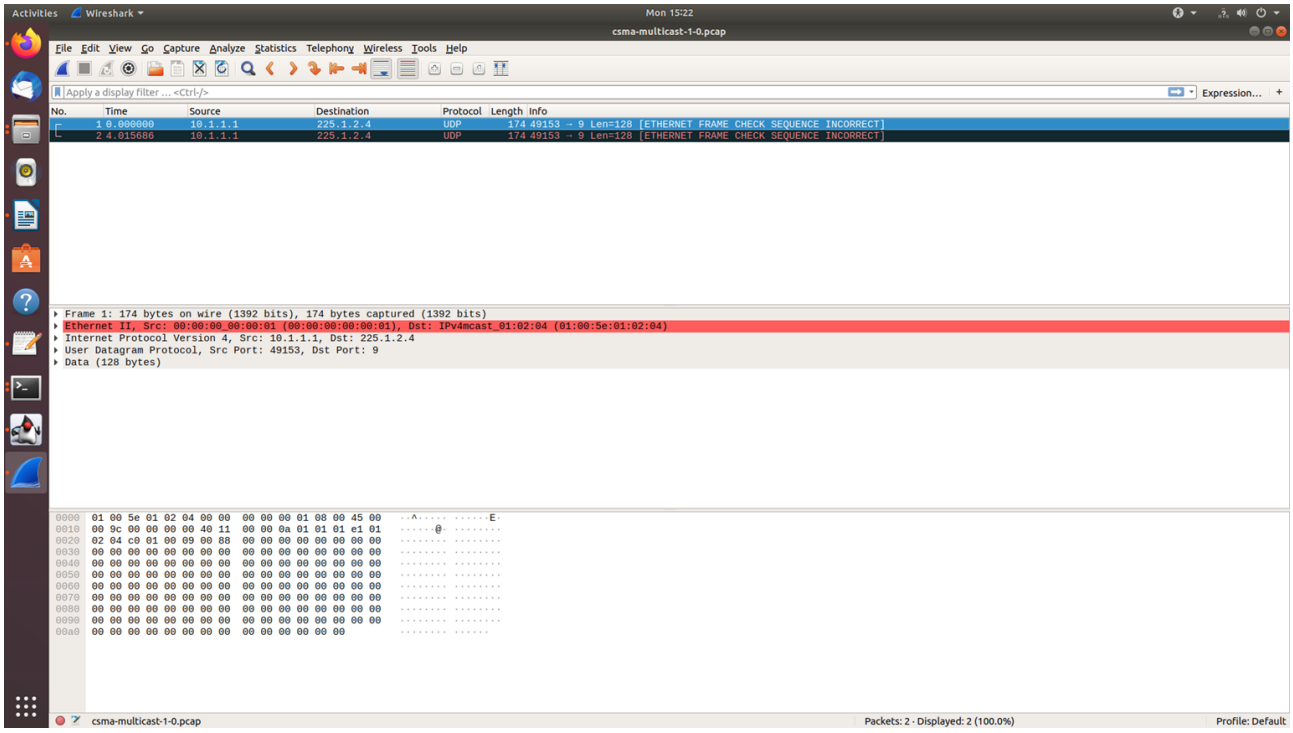




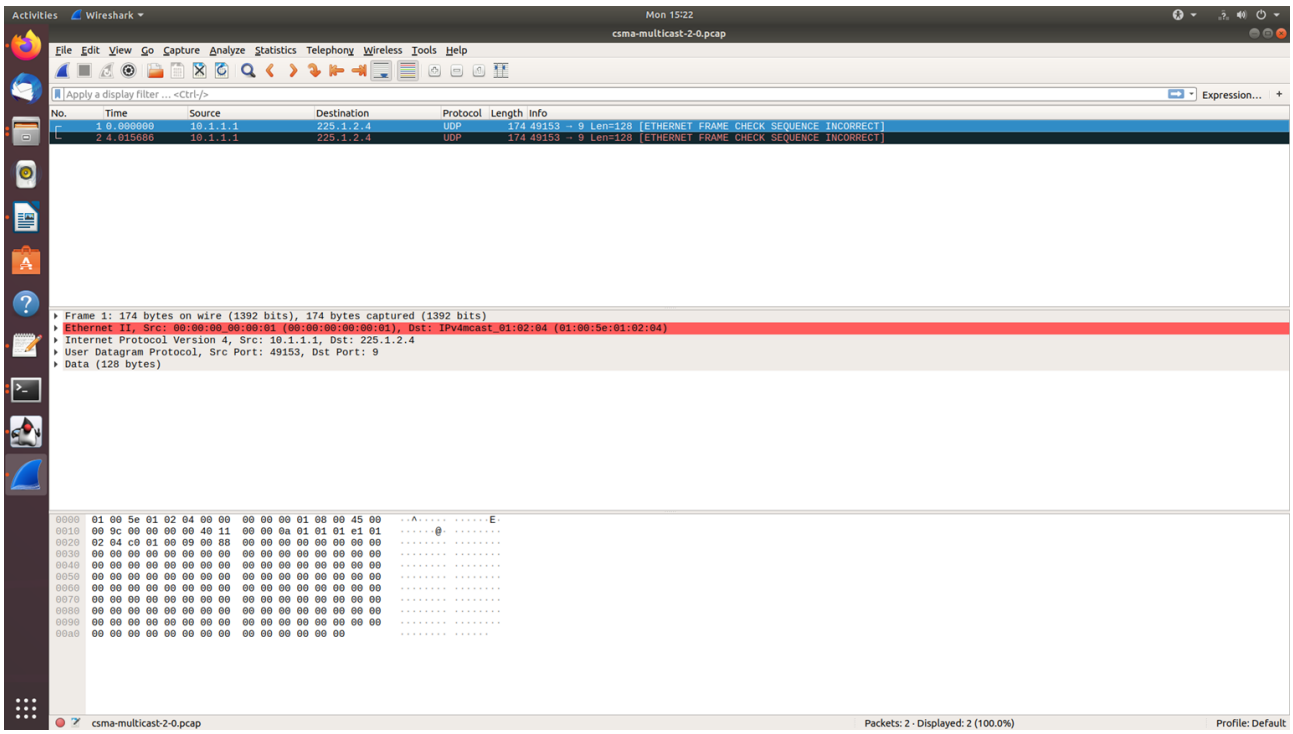
Wireshark:

NS LAB 28 AUGUST | 120CS0124 Maloth Aditya

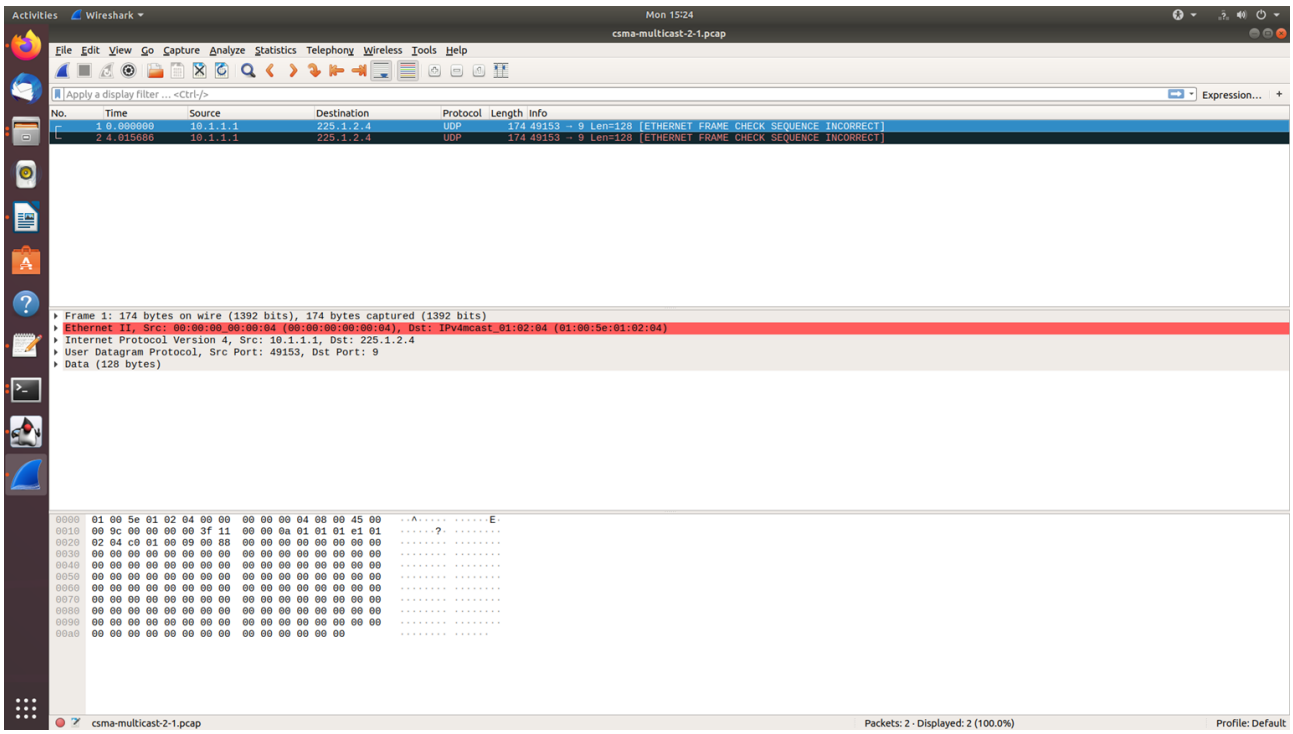


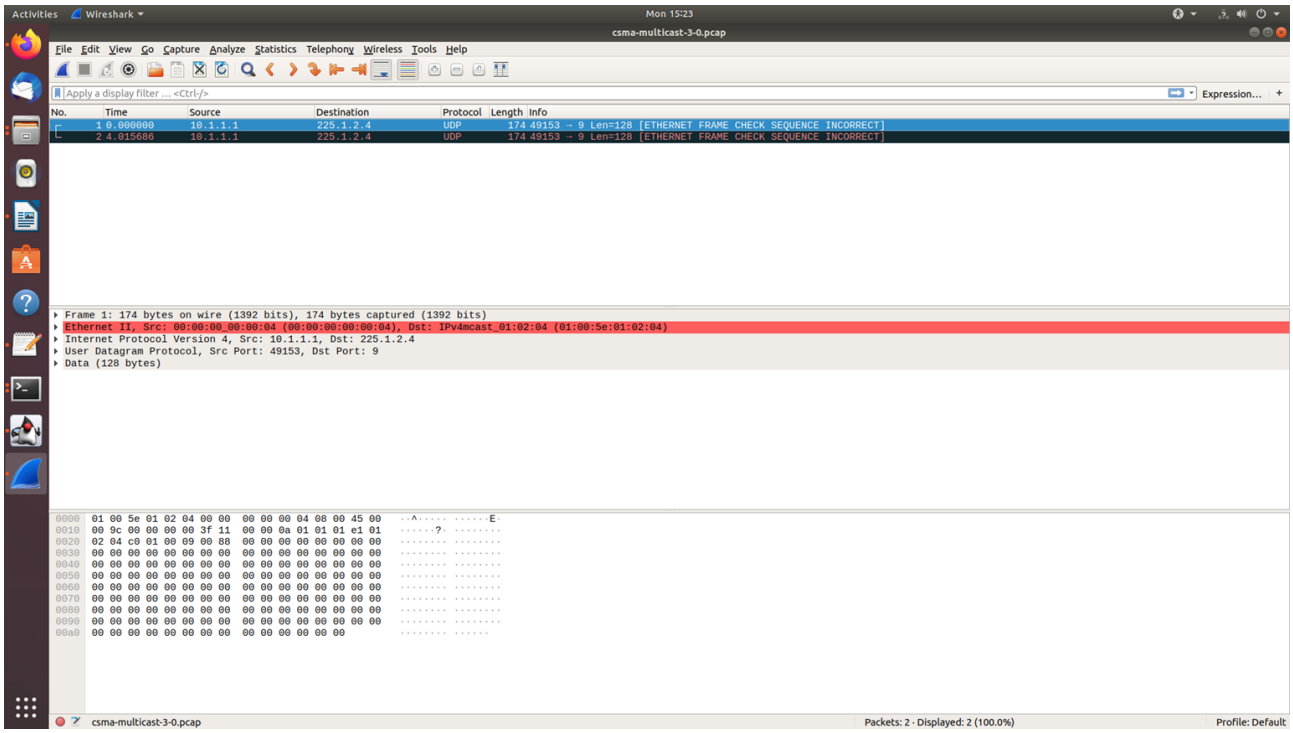


NS LAB 28 AUGUST | 120CS0124 Maloth Aditya

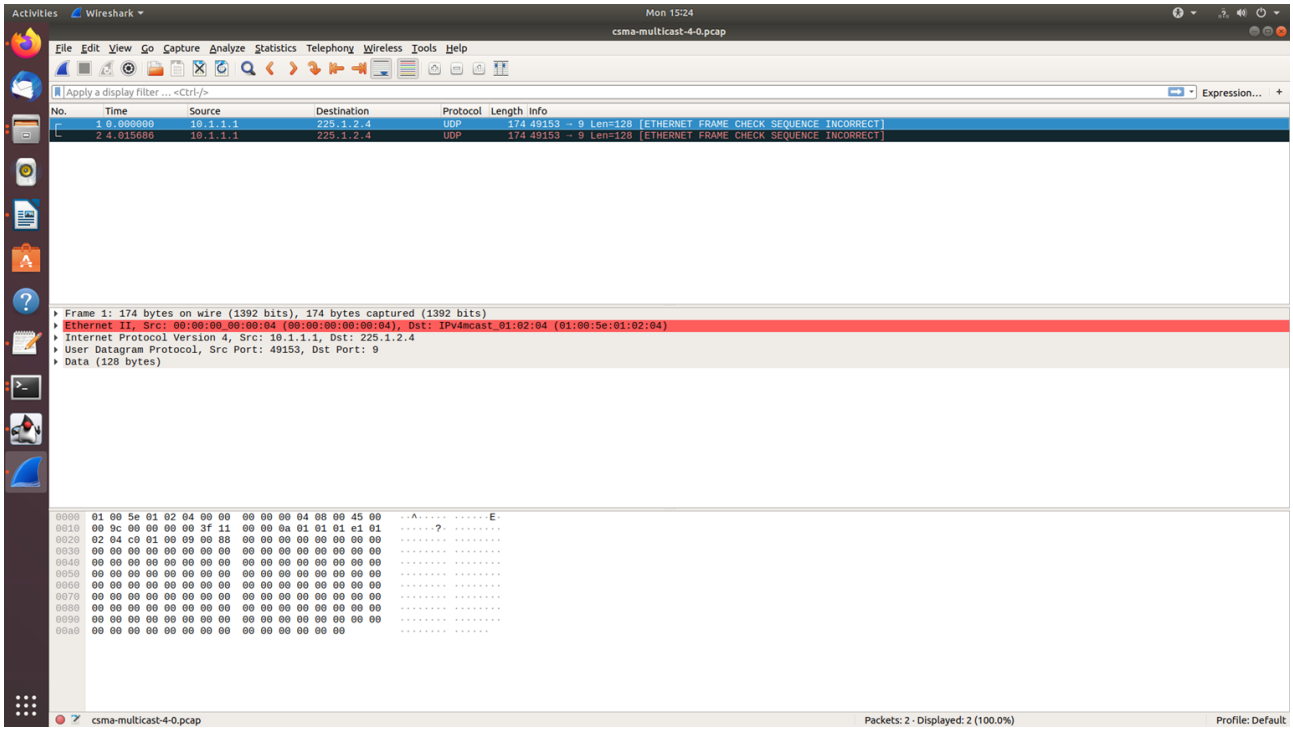


NS LAB 28 AUGUST | 120CS0124 Maloth Aditya





NS LAB 28 AUGUST | 120CS0124 Maloth Aditya



CSMA-one-subnet CODE:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
 /*
  * 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
  */

 // Network topology
 //
```

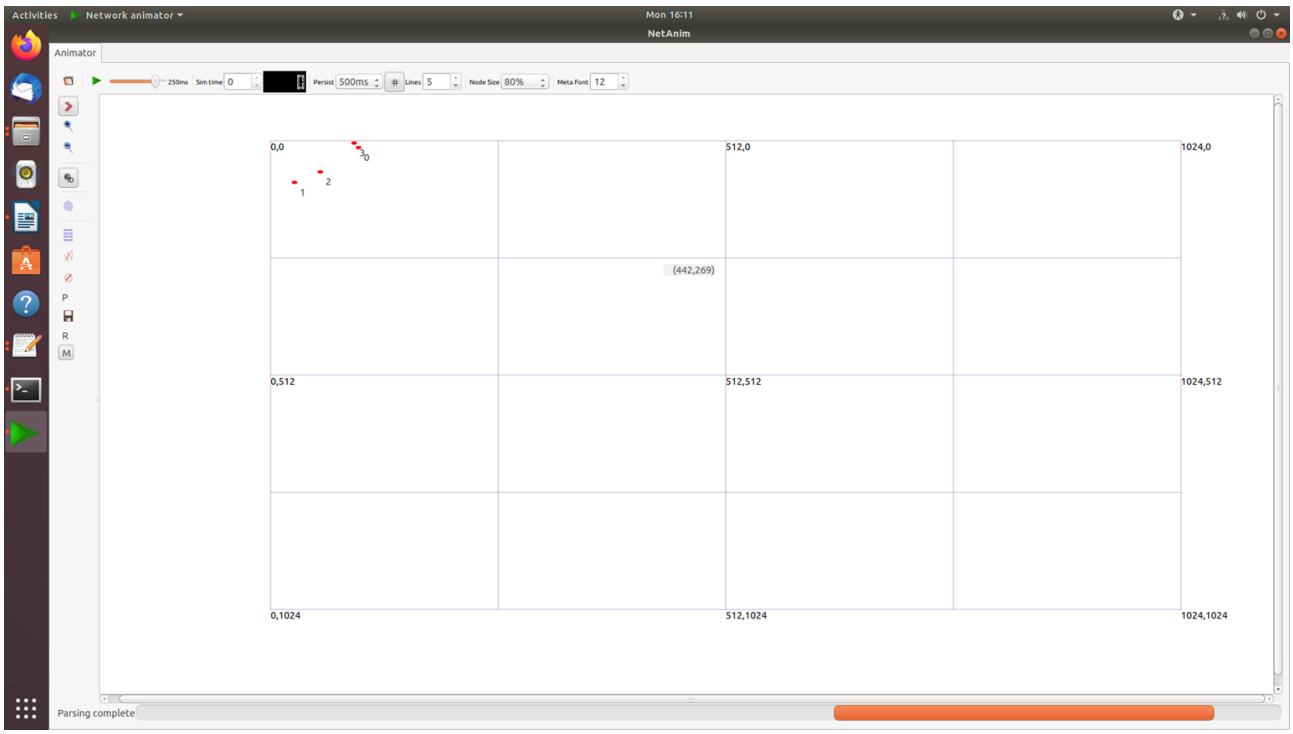
```
//      n0   n1   n2   n3
//      |    |    |    |
//      =================
//            LAN
//
// - CBR/UDP flows from n0 to n1 and from n3 to n0
// - DropTail queues
// - Tracing of queues and packet receptions to file "csma-one-subnet.tr"

#include <iostream>
#include <fstream>

#include "ns3/core-module.h"
 #include "ns3/network-module.h"
 #include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/netanim-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("CsmaOneSubnetExample");

int
main (int argc, char *argv[])
{
//
// Users may find it convenient to turn on explicit debugging
// for selected modules; the below lines suggest how to do this
//
#if 0
  LogComponentEnable ("CsmaOneSubnetExample", LOG_LEVEL_INFO);
#endif
//
// Allow the user to override any of the defaults and the above Bind() at
// run-time, via command-line arguments
//
  CommandLine cmd;
  cmd.Parse (argc, argv);
//
// Explicitly create the nodes required by the topology (shown above).
//
  NS_LOG_INFO ("Create nodes.");
  NodeContainer nodes;
  nodes.Create (4);

  NS_LOG_INFO ("Build Topology");
  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", DataRateValue (5000000));
```

```
  csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
//
// Now fill out the topology by creating the net devices required to connect
// the nodes to the channels and hooking them up.
//
  NetDeviceContainer devices = csma.Install (nodes);

  InternetStackHelper internet;
  internet.Install (nodes);

// We've got the "hardware" in place.  Now we need to add IP addresses.
//
  NS_LOG_INFO ("Assign IP Addresses.");
  Ipv4AddressHelper ipv4;
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer interfaces = ipv4.Assign (devices);

//
// Create an OnOff application to send UDP datagrams from node zero to node 1.
//
  NS_LOG_INFO ("Create Applications.");
  uint16_t port = 9;   // Discard port (RFC 863)

  OnOffHelper onoff ("ns3::UdpSocketFactory",
                     Address (InetSocketAddress (interfaces.GetAddress (1), port)));
  onoff.SetConstantRate (DataRate ("500kb/s"));

  ApplicationContainer app = onoff.Install (nodes.Get (0));
  // Start the application
  app.Start (Seconds (1.0));
  app.Stop (Seconds (10.0));

  // Create an optional packet sink to receive these packets
  PacketSinkHelper sink ("ns3::UdpSocketFactory",
                         Address (InetSocketAddress (Ipv4Address::GetAny (), port)));
  app = sink.Install (nodes.Get (1));
  app.Start (Seconds (0.0));

//
// Create a similar flow from n3 to n0, starting at time 1.1 seconds
//
  onoff.SetAttribute ("Remote",
                      AddressValue (InetSocketAddress (interfaces.GetAddress (0), port)));
  app = onoff.Install (nodes.Get (3));
  app.Start (Seconds (1.1));
  app.Stop (Seconds (10.0));

  app = sink.Install (nodes.Get (0));
  app.Start (Seconds (0.0));
```

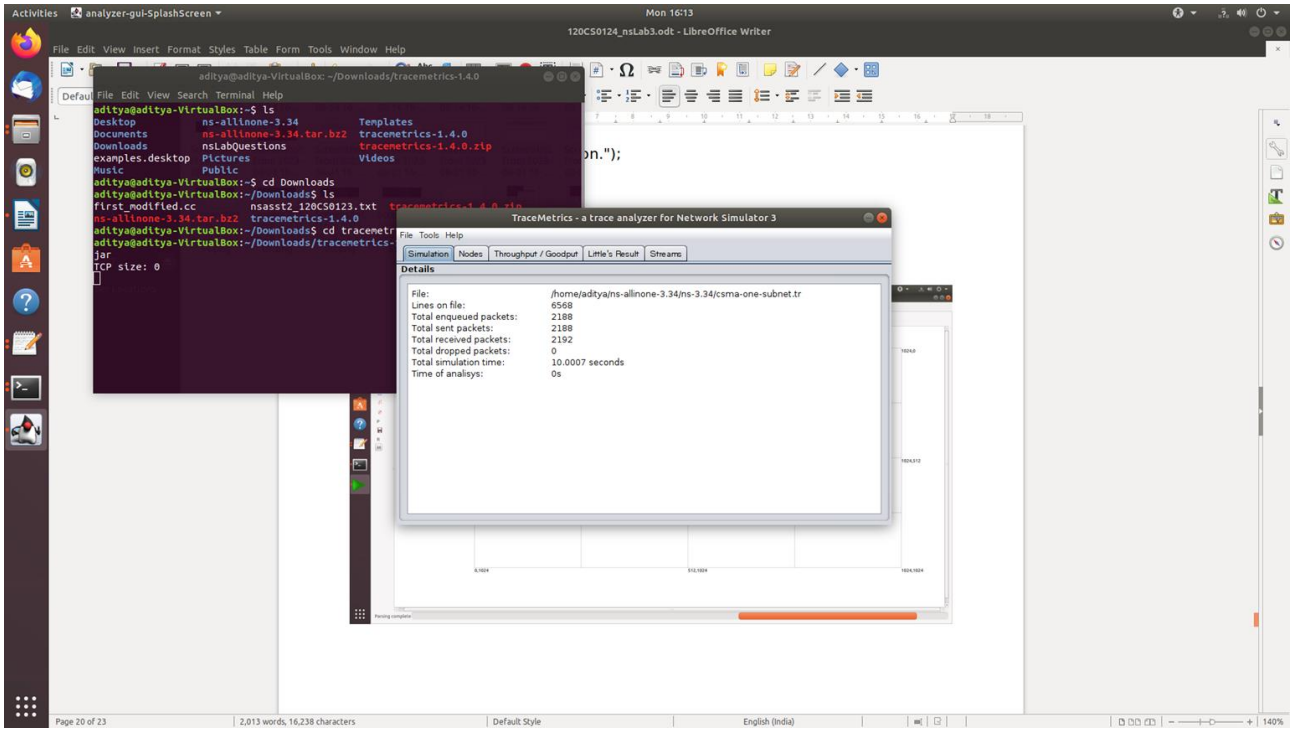
```
  NS_LOG_INFO ("Configure Tracing.");
 //
 // Configure ascii tracing of all enqueue, dequeue, and NetDevice receive
 // events on all devices.  Trace output will be sent to the file
 // "csma-one-subnet.tr"
 //


AnimationInterface anim("csma-one-subnet.xml");
anim.SetConstantPosition(nodes.Get(0), 10.0, 40.0);
anim.SetConstantPosition(nodes.Get(1), 20.0, 30.0);
anim.SetConstantPosition(nodes.Get(2), 30.0, 20.0);
anim.SetConstantPosition(nodes.Get(3), 40.0, 10.0);

AsciiTraceHelper ascii;
  csma.EnableAsciiAll (ascii.CreateFileStream ("csma-one-subnet.tr"));

 //
 // Also configure some tcpdump traces; each interface will be traced.
 // The output files will be named:
 //
 //     csma-one-subnet-<node ID>-<device's interface index>.pcap
 //
 // and can be read by the "tcpdump -r" command (use "-tt" option to
 // display timestamps correctly)
 //
  csma.EnablePcapAll ("csma-one-subnet", false);
 //
 // Now, do the actual simulation.
 //
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
}
```

OUTPUT:
NetAnim:

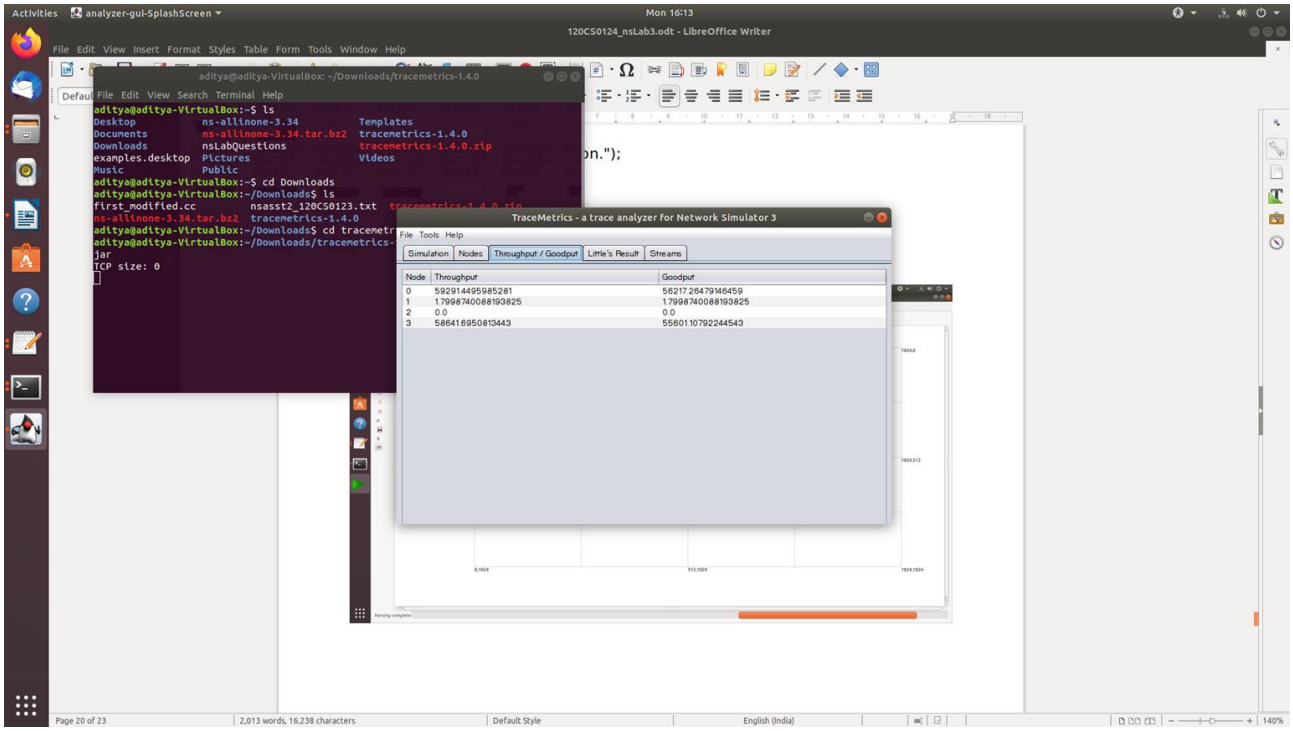
NS LAB 28 AUGUST | 120CS0124 Maloth Aditya



Tracemetrics:

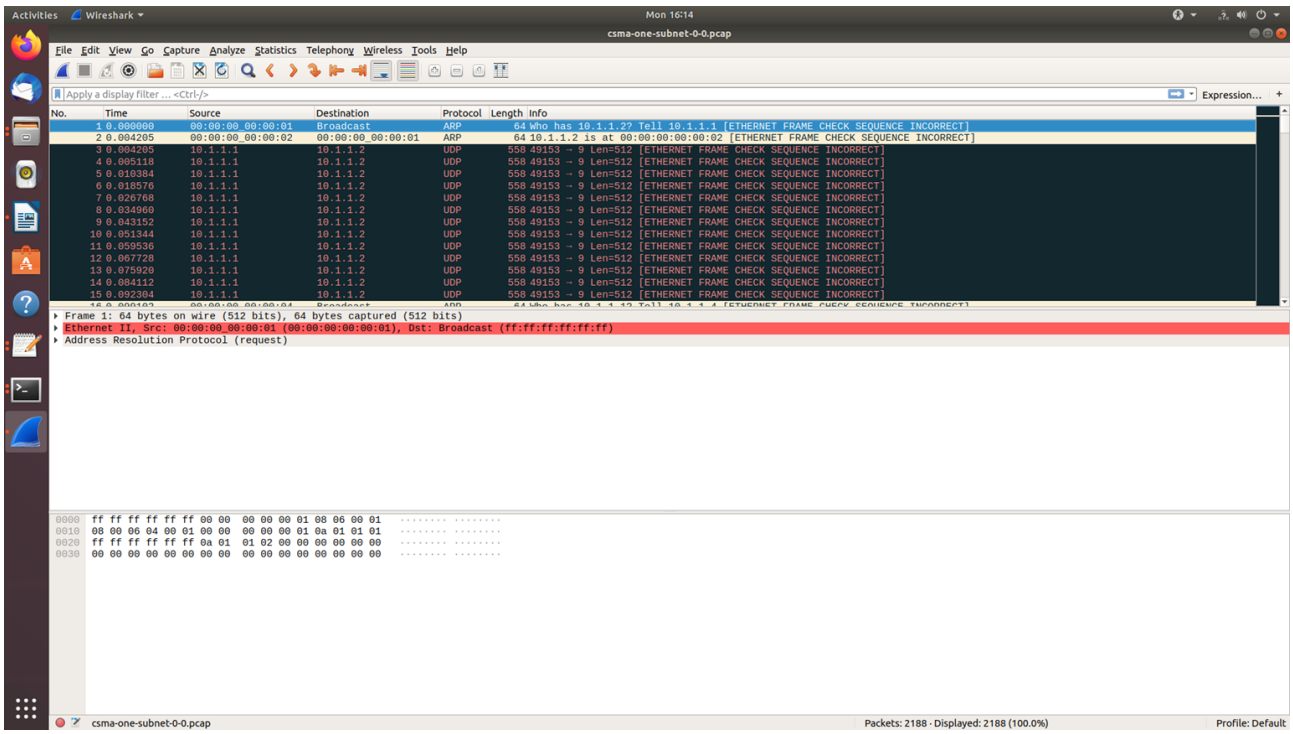


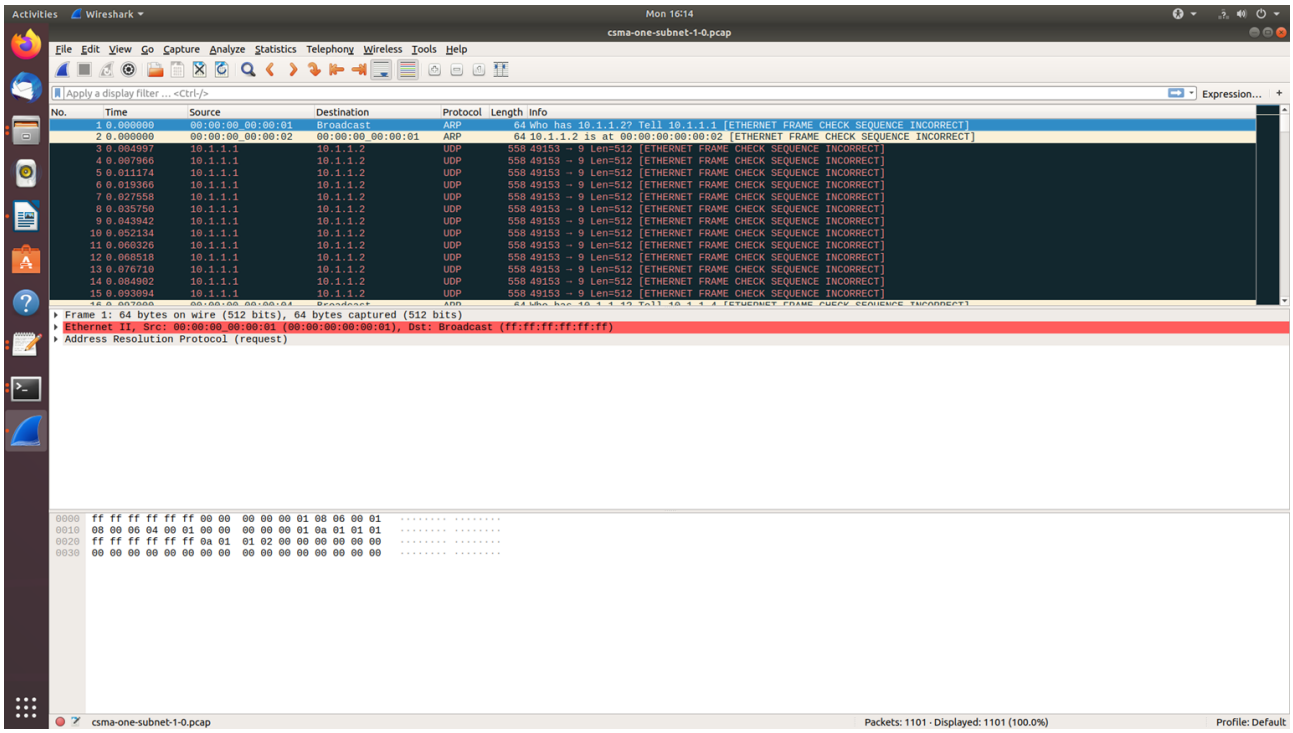
NS LAB 28 AUGUST | 120CS0124 Maloth Aditya



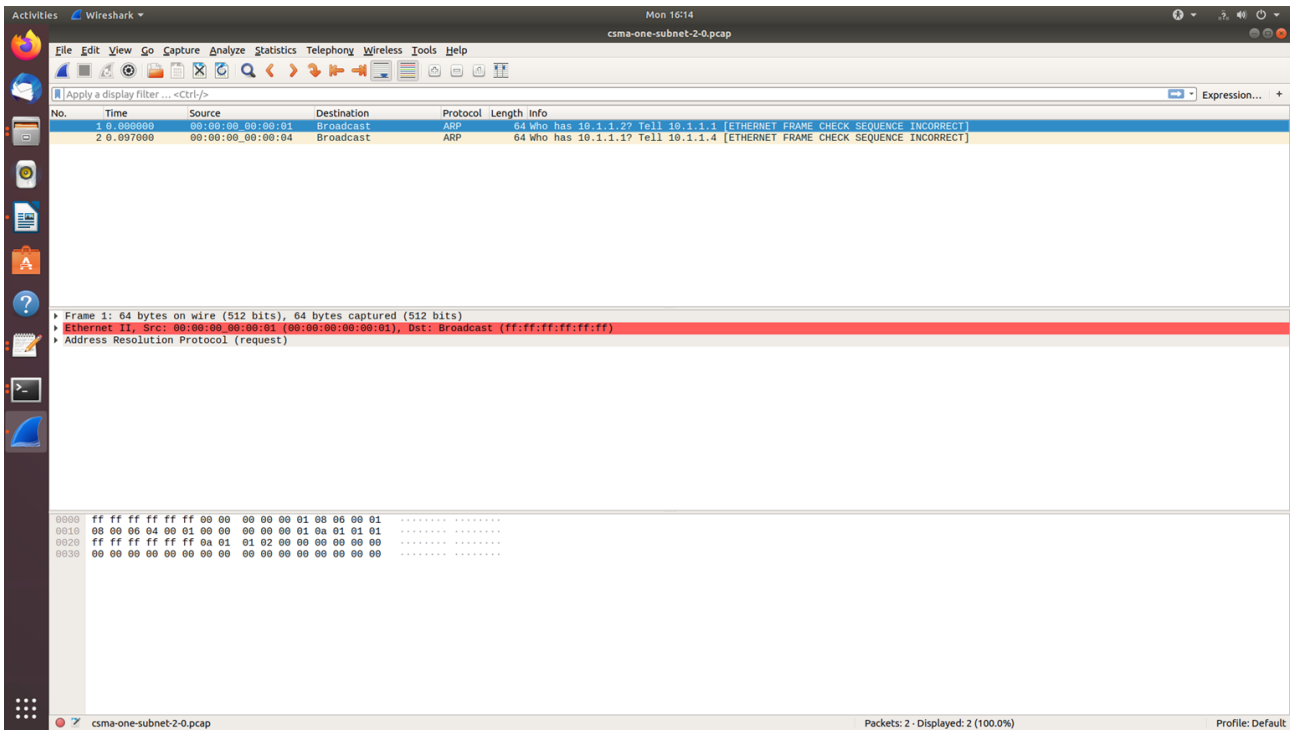
Wireshark:

NS LAB 28 AUGUST | 120CS0124 Maloth Aditya





NS LAB 28 AUGUST | 120CS0124 Maloth Aditya



NS LAB 28 AUGUST | 120CS0124 Maloth Aditya