Simulations of CSMA-Multicast using NS3

Gopika S Pai

Abstract

Multicast is a group communication methodology whre information is addressed to a group of destination computers simultaneously. Multicasting can allow you to reach many end users while utilizing only a single data stream, decreasing bandwidth and resource utilization. CSMA(Carrier Sense Multiple Access)is an Ethernet(IEEE 802.3) MAC protocol for carrier transmission. On Ethernet each device can attempt to send a frame anytime. Each device senses whether the line is idle by sending an RTS (Ready to Send) frame and and thereby confirming the line is clear to send via reception of a CTS (Clear to Send) frame. Here, in each CSMA Multicast scenario, I have tried to implement different topologies in Multicast networks considering CSMA protocol.

Contents

1 Introduction

There are three fundamental types of IPv4 addresses: 1) Unicast : A unicast address is designed to transmit a packet to a single destination. 2) Broadcast : A broadcast address is used to send a datagram to an entire subnetwork. 3) Multicast : A multicast address is designed to enable the delivery of datagrams to a set of hosts that have been configured as members of a multicast group in various scattered subnetworks.

Here, I have created simulations on the multicast addressing. There are four different topologies on CSMA-Multicast and UDP is been used in all the topologies.

1.1 Multicast

Multicast is group communication where information is addressed to a group of destination computers simultaneously.It is similar to broadcast in the sense it's target is a number of machines in the network, but not all.Multicast is only directed to a group of hosts in the network.The most common transport layer protocol to use multicast addressing is User Datagram Protocol (UDP). By its nature, UDP is not reliable—messages may be lost or delivered out of order.

Multicast networks are used in softwares, such as

1. Whatsapp
2. Sending Email
3. Teleconferencing
4. Videoconferencing

   and many more.

1.2 NS3(Network Simulator 3)

It is a free software, publicly available under the GNU GPLv2 license for research and development.

2 Motivation

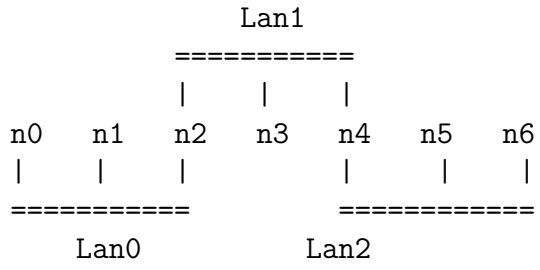
Multicast is the most cost effective way to send pictures or data in a bulk from any one point to any multitude of receivers. Broadcast or Unicast is comparatively expensive than Multicast, as it needs a one to one broadcast-receiver relationship like a telephone connection.Multicast is more efficient than broadcast, because broadcast packets have to be received by everyone on the local link. Each OS takes an interrupt, and passes the packet on for inspection, which normally involves some data copies. In multicast, the network card doesn't listen to these multicast packets unless it has been told to do so. On September 11, 2001, duing the attacks the Internet had suffered several interesting failures.The giant new channels and many other nes related sites crashed due to high traffic. If those servers and their distribution infrastructure had been IP Multicast this wouldn't have happened. Multicast selects in who it sends to and what. If someone wants to stream into the news channels we don't want every router on the Internet consume the bandwidth required to deliver it to each computer.If nobody wants to see it within your network, there's no reason to let it travel into the network.

Below are few scenarios based on Multicast.

3 Simulation scenarios

3.1 Scenario 1

Desired Topology:



3.1.1 Brief discription

- Multicast source is at node n0.
- Multicast is forwarded by node n2 to LAN1 and by n4 to LAN2
- Nodes n0, n1, n2, n3, n4, n5 and n6 receive the multicast frame.
- Node n6 listens for the data

- The actual code begins by loading module include files.

```
#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"
```

- The ns-3 namespace is used and a logging component is defined.

```
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("CsmaMulticastExample");
```

- Starting with the main program

```
int
main (int argc, char *argv[])
{
```

- For the users to find it convenient to turn on explicit debugging for selected modules

```
LogComponentEnable ("CsmaMulticastExample", LOG_LEVEL_INFO);
```

- Selecting the encapsulation mode

```
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);
```

- Creating seven nodes for the topology

```
NS_LOG_INFO ("Create nodes.");
NodeContainer c;
c.Create (7);
```

- We will later want three subcontainers for these nodes, for the three 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));
NodeContainer c2 = NodeContainer (c.Get (4), c.Get (5), c.Get (6));
```

```
NS_LOG_INFO ("Build Topology.");
CsmaHelper csma;
```

- Setting the channel attributes data flow rate and time

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

- We are installing all the nodes to that LANs ,c0 to first ,c1 to second and c2 to third respectively The NetDeviceContainers created here will be used later, for IP addressing

```
NetDeviceContainer nd0 = csma.Install (c0);  // First LAN
NetDeviceContainer nd1 = csma.Install (c1);  // Second LAN
NetDeviceContainer nd2 = csma.Install (c2);  // Third LAN
```

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

- Assigning the IP Addresses

```
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);
ipv4Addr.SetBase ("10.1.3.0", "255.255.255.0");
ipv4Addr.Assign (nd2);
```

```
NS_LOG_INFO ("Configure multicasting.");
```

- *Now we can configure multicasting. As described above, the multicast source is at node zero, for which we assign the IP address as 10.1.1.1 We need to define a multicast group to send packets too. 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:

- Configure a (static) multicast route on node n2
- Set up a default multicast route on the sender n0
- Have node n6 join the multicast group We have a helper that can help us with static multicast

```
Ipv4StaticRoutingHelper multicast;
```

- Configure a (static) multicast route on node n4 (multicastRouter)

Node 2 acts as a router

```
Ptr<Node> multicastRouter = c.Get (2);
```

nd0: NetDeviceContainer 0 i.e LAN1, where n2 acts as an input interface for LAN2

```
Ptr<NetDevice> inputIf = nd0.Get (2);
```

A container of output NetDevices (we only need one NetDevice here)

```
NetDeviceContainer outputDevices;
outputDevices.Add (nd1.Get (0));
multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);
```

Node 4 acts as a router

```
Ptr<Node> multicastRouter = c.Get (4);
```

nd1: NetDeviceContainer 1 i.e LAN2, where n4 acts as an input interface for LAN3

```
Ptr<NetDevice> inputIf = nd1.Get (4);
```

A container of output NetDevices (we only need one NetDevice here)

```
NetDeviceContainer outputDevices;
outputDevices.Add (nd2.Get (0));
multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);
```

- Set up a default multicast route on the sender n0

Node 0 acts as the sender

```
Ptr<Node> sender = c.Get (0);
```

In LAN0 Node 0 acts as the sender interface to LAN0

```
Ptr<NetDevice> senderIf = nd0.Get (0);
multicast.SetDefaultMulticastRoute (sender, senderIf);
```

- Set up a default multicast route on the sender n2

Node 2 acts as the sender

```
Ptr<Node> sender1 = c.Get (2);
```

In LAN0 Node 2 acts as the sender interface to LAN1

```
 Ptr<NetDevice> senderIf1 = nd1.Get (0);
multicast.SetDefaultMulticastRoute (sender1, senderIf1);
```

- Set up a default multicast route on the sender n4

Node 4 acts as the sender

```
 Ptr<Node> sender2 = c.Get (4);
```

In LAN1 Node 4 acts as the sender interface to LAN2

```
Ptr<NetDevice> senderIf2 = nd2.Get (0);
multicast.SetDefaultMulticastRoute (sender2, senderIf2);
```

- Create an OnOff application to send UDP datagrams from node zero to the multicast group (node six will be listening).

```
NS_LOG_INFO ("Create Applications.");
```

The port no is changed from the default (RFC 863) to 9

```
uint16_t multicastPort = 9;
```

- Configure a multicast packet generator that generates a packet every few seconds
- Onoff acts as when to send the packets and when to block

```
OnOffHelper onoff ("ns3::UdpSocketFactory",
Address (InetSocketAddress (multicastGroup, multicastPort)));
```

- Setting up the data flow rate

```
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 n2
sinkC.Start (Seconds (1.0));
sinkC.Stop (Seconds (12.0));
```

- 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", true);
```

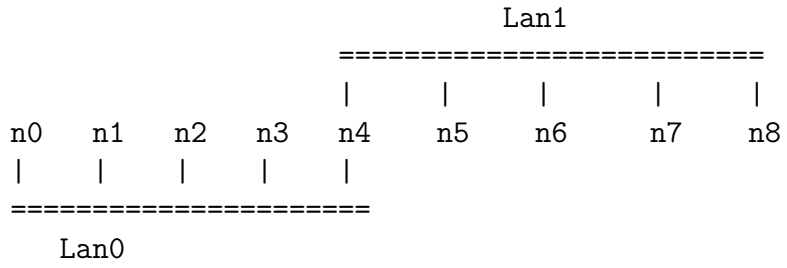
- Now, we do the actual simulation.

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

}
```

3.2 Scenario 2

Desired Topology:



3.2.1 Brief discription

- Multicast source is at node n0.
- Multicast forwarded by node n4 to LAN1
- Nodes n0, n1, n2, n3, n4, n5, n6, n7 and n8 receive the multicast frame.
- Node n8 listens for the data

- A LAN cannot hold more than 5 node at a time.

(Refer `Scenario 1` for the following code as it uses the same methodology.)

- Creating 9 nodes for the topology

```
NS_LOG_INFO ("Create nodes.");
NodeContainer c;
c.Create (9);
```

- 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),
                                  c.Get (3), c.Get (4));
NodeContainer c1 = NodeContainer (c.Get (4), c.Get (5), c.Get (6),
                                  c.Get (7), c.Get (8));
```

```
NS_LOG_INFO ("Build Topology.");
CsmaHelper csma;
```

- Setting the channel attributes data flow rate and time

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

- We are installing all the nodes to that LANs ,c0 to first and c1 to second The NetDeviceContainers created here will be used 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);
```

- Assigning the IP Addresses

```
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, for which we assign the IP address as 10.1.1.1 We need to define a multicast group to send packets too. 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:

- Configure a (static) multicast route on node n2
- Set up a default multicast route on the sender n0
- Have node n8 join the multicast group We have a helper that can help us with static multicast

```
Ipv4StaticRoutingHelper multicast;
```

- Configure a (static) multicast route on node n8 (multicastRouter)

Node 4 acts as a router

```
Ptr<Node> multicastRouter = c.Get (4);
```

nd0: NetDeviceContainer 0 i.e LAN1, where n4 acts as an input interface for LAN2

```
Ptr<NetDevice> inputIf = nd0.Get (4);
```

A container of output NetDevices (we only need one NetDevice here)

```
NetDeviceContainer outputDevices;
outputDevices.Add (nd1.Get (0));
multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);
```

- Set up a default multicast route on the sender n0

Node 0 acts as the sender

```
Ptr<Node> sender = c.Get (0);
```

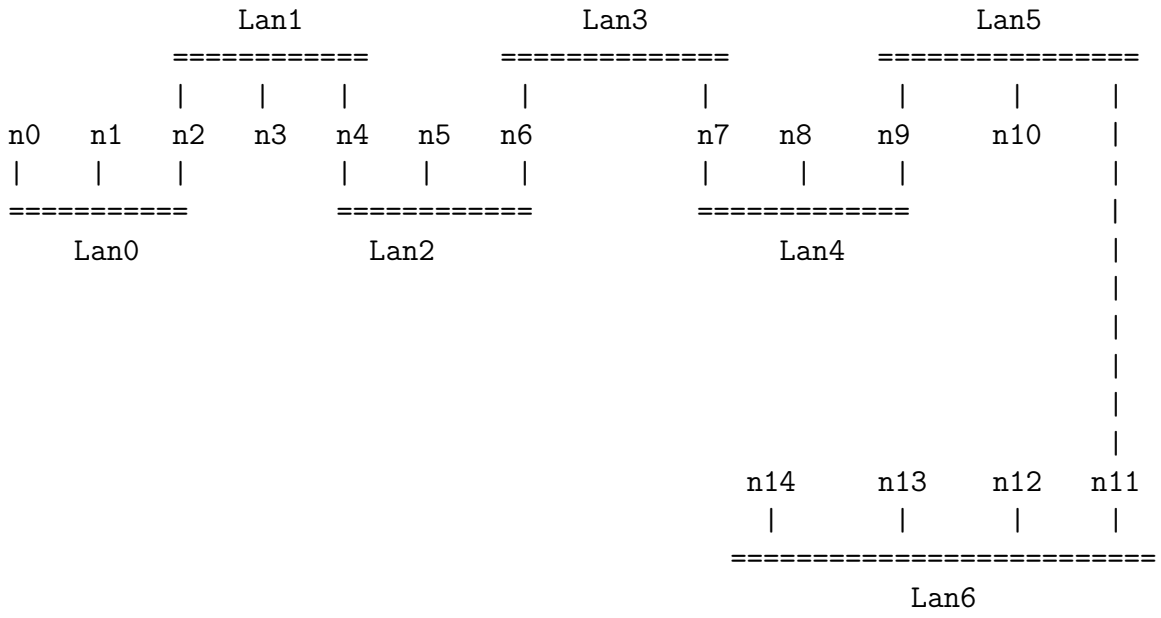
In LAN0 Node 0 acts as the sender interface to LAN0

```
  Ptr<NetDevice> senderIf = nd0.Get (0);
multicast.SetDefaultMulticastRoute (sender, senderIf);
```

(Refer `Scenario 1` for the following code as it uses the same methodology.)

3.3 Scenario 3

Desired Topology:



3.3.1 Brief discription

- Multicast source is at node n0.
- Multicast forwarded by node n2 to LAN1, n4 to LAN2, n6 to LAN3, n7 to LAN4, n9 to LAN5 and n11 to LAN6
- Nodes n0, n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14 receive the multicast frame.
- Node n14 listens for the data

(Refer `Scenario 1` for the following code as it uses the same methodology.)

- Creating 15 nodes for the topology

```
NS_LOG_INFO ("Create nodes.");
NodeContainer c;
c.Create (15);
```

- We will later want seven subcontainers for these nodes, for the seven LANs, each with different number of nodes

```
NodeContainer c0 = NodeContainer (c.Get (0), c.Get (1), c.Get (2));
NodeContainer c1 = NodeContainer (c.Get (2), c.Get (3), c.Get (4));
NodeContainer c2 = NodeContainer (c.Get (4), c.Get (5), c.Get (6));
NodeContainer c3 = NodeContainer (c.Get (6), c.Get (7));
NodeContainer c4 = NodeContainer (c.Get (7), c.Get (8), c.Get (9));
NodeContainer c5 = NodeContainer (c.Get (9), c.Get (10), c.Get (11));
NodeContainer c6 = NodeContainer (c.Get (11), c.Get (12),
                                  c.Get (13), c.Get(14));


NS_LOG_INFO ("Build Topology.");
CsmaHelper csma;
```

- Setting the channel attributes data flow rate and time

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

- We are installing all the nodes to that LANs ,c0 to first, c1 to second, c2 to third, c3 to fourth, c4 to fifth and c5 to sixth and c6 to seventh The NetDeviceContainers created here will be used later, for IP addressing

```
NetDeviceContainer nd0 = csma.Install (c0);  // First LAN
NetDeviceContainer nd1 = csma.Install (c1);  // Second LAN
NetDeviceContainer nd2 = csma.Install (c2);  // Third LAN
NetDeviceContainer nd3 = csma.Install (c3);  // fourth LAN
NetDeviceContainer nd4 = csma.Install (c4);  // Fifth LAN
NetDeviceContainer nd5 = csma.Install (c5);  // Sixth LAN
NetDeviceContainer nd6 = csma.Install (c6);  // Seventh LAN


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

- Assigning the IP Addresses

```
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);
ipv4Addr.SetBase ("10.1.3.0", "255.255.255.0");
ipv4Addr.Assign (nd2);
ipv4Addr.SetBase ("10.1.4.0", "255.255.255.0");
ipv4Addr.Assign (nd3);
ipv4Addr.SetBase ("10.1.5.0", "255.255.255.0");
ipv4Addr.Assign (nd4);
ipv4Addr.SetBase ("10.1.6.0", "255.255.255.0");
ipv4Addr.Assign (nd5);
ipv4Addr.SetBase ("10.1.7.0", "255.255.255.0");
ipv4Addr.Assign (nd6);
```

```
NS_LOG_INFO ("Configure multicasting.");
```

- *Now we can configure multicasting. As described above, the multicast source is at node zero, for which we assign the IP address as 10.1.1.1 We need to define a multicast group to send packets too. 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:

- Configure a (static) multicast route on node n2, n4, n6 , n7, n9,n11
- Set up a default multicast route on the sender n0,n2,n4,n6,n7,n9,n11
- Have node n11 join the multicast group We have a helper that can help us with static multicast

```
Ipv4StaticRoutingHelper multicast;
```

- Configure a (static) multicast route on node n2 (multicastRouter)

Node 2 acts as a router

```
Ptr<Node> multicastRouter = c.Get (2);
```

nd0: NetDeviceContainer 0 i.e LAN0, where n2 acts as an input interface for LAN2

```
Ptr<NetDevice> inputIf = nd0.Get (2);
```

A container of output NetDevices (we only need one NetDevice here)

```
NetDeviceContainer outputDevices;
outputDevices.Add (nd1.Get (0));
multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);
```

- Configure a (static) multicast route on node n4 (multicastRouter)

Node 4 acts as a router

```
Ptr<Node> multicastRouter = c.Get (4);
```

nd1: NetDeviceContainer 1 i.e LAN1, where n4 acts as an input interface for LAN2

```
Ptr<NetDevice> inputIf = nd1.Get (4);
```

A container of output NetDevices (we only need one NetDevice here)

```
NetDeviceContainer outputDevices;
outputDevices.Add (nd2.Get (0));
multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);
```

- Configure a (static) multicast route on node n6 (multicastRouter)

Node 6 acts as a router

```
Ptr<Node> multicastRouter = c.Get (6);
```

nd2: NetDeviceContainer 2 i.e LAN2, where n6 acts as an input interface for LAN3

```
Ptr<NetDevice> inputIf = nd2.Get (6);
```

A container of output NetDevices (we only need one NetDevice here)

```
NetDeviceContainer outputDevices;
outputDevices.Add (nd3.Get (0));
multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);
```

- Set up a default multicast route on the sender n0
- Configure a (static) multicast route on node n7 (multicastRouter)

Node 7 acts as a router

```
Ptr<Node> multicastRouter = c.Get (7);
```

nd3: NetDeviceContainer 3 i.e LAN4, where n6 acts as an input interface for LAN5

```
Ptr<NetDevice> inputIf = nd3.Get (7);
```

A container of output NetDevices (we only need one NetDevice here)

```
NetDeviceContainer outputDevices;
outputDevices.Add (nd4.Get (0));
multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);
```

- Configure a (static) multicast route on node n9 (multicastRouter)

Node 9 acts as a router

```
Ptr<Node> multicastRouter = c.Get (9);
```

nd4: NetDeviceContainer 4 i.e LAN4, where n9 acts as an input interface for LAN5

```
Ptr<NetDevice> inputIf = nd4.Get (9);
```

A container of output NetDevices (we only need one NetDevice here)

```
NetDeviceContainer outputDevices;
outputDevices.Add (nd5.Get (0));
multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);
```

- Configure a (static) multicast route on node n11 (multicastRouter)

Node 11 acts as a router

```
Ptr<Node> multicastRouter = c.Get (11);
```

nd5: NetDeviceContainer 5 i.e LAN5, where n11 acts as an input interface for LAN6

```
Ptr<NetDevice> inputIf = nd5.Get (11);
```

A container of output NetDevices (we only need one NetDevice here)

```
NetDeviceContainer outputDevices;
outputDevices.Add (nd6.Get (0));
multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);
```

Node 0 acts as the sender

```
Ptr<Node> sender = c.Get (0);
```

In LAN0 Node 0 acts as the sender interface to LAN0

```
Ptr<NetDevice> senderIf = nd0.Get (0);
multicast.SetDefaultMulticastRoute (sender, senderIf);
```

Node 2 acts as the sender

```
Ptr<Node> sender1 = c.Get (2);
```

In LAN0 Node 2 acts as the sender interface to LAN1

```
Ptr<NetDevice> senderIf1 = nd1.Get (2);
multicast.SetDefaultMulticastRoute (sender1, senderIf1);
```

Node 4 acts as the sender

```
Ptr<Node> sender2 = c.Get (4);
```

In LAN1 Node 4 acts as the sender interface to LAN2

```
  Ptr<NetDevice> senderIf2 = nd2.Get (4);
multicast.SetDefaultMulticastRoute (sender2, senderIf2);
```

Node 6 acts as the sender

```
Ptr<Node> sender3 = c.Get (6);
```

In LAN2 Node 6 acts as the sender interface to LAN3

```
  Ptr<NetDevice> senderIf3 = nd3.Get (6);
multicast.SetDefaultMulticastRoute (sender3, senderIf3);
```

Node 7 acts as the sender

```
Ptr<Node> sender4 = c.Get (7);
```

In LAN3 Node 7 acts as the sender interface to LAN4

```
  Ptr<NetDevice> senderIf4 = nd4.Get (7);
multicast.SetDefaultMulticastRoute (sender4, senderIf4);
```

Node 9 acts as the sender

```
Ptr<Node> sender5 = c.Get (9);
```

In LAN4 Node 9 acts as the sender interface to LAN5

```
  Ptr<NetDevice> senderIf5 = nd5.Get (9);
multicast.SetDefaultMulticastRoute (sender5, senderIf5);
```

Node 11 acts as the sender

```
Ptr<Node> sender6 = c.Get (11);
```

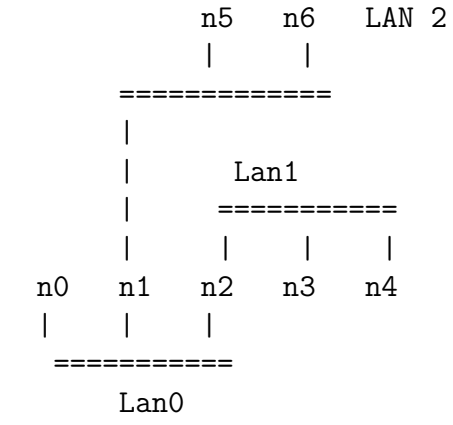
In LAN5 Node 11 acts as the sender interface to LAN6

```
  Ptr<NetDevice> senderIf6 = nd6.Get (11);
multicast.SetDefaultMulticastRoute (sender6, senderIf6);
```

(Refer `Scenario 1` for the following code as it uses the same methodology.)

3.4 Scenario 4

Desired Topology:



3.4.1 Brief discription

- Multicast source is at node n0.
- Multicast forwarded by node n2 to LAN1 and n1 to LAN2
- Nodes n0, n1, n2, n3, n4, n5,and n6 receive the multicast frame.
- Node n6 and n4 listens for the data

(Refer `Scenario 1` for the following code as it uses the same methodology.)

- Creating 7 nodes for the topology

```
NS_LOG_INFO ("Create nodes.");
NodeContainer c;
c.Create (7);
```

- We will later want three subcontainers of these nodes, for the three LANs

```
NodeContainer c0 = NodeContainer (c.Get (0), c.Get (1), c.Get (2));
  NodeContainer c2 = NodeContainer (c.Get (1), c.Get (5), c.Get (6));
  NodeContainer c1 = NodeContainer (c.Get (2), c.Get (3), c.Get (4));
```

```
  NS_LOG_INFO ("Build Topology.");
  CsmaHelper csma;
```

- Setting the channel attributes data flow rate and time

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

- We are installing all the nodes to that LANs ,c0 to first and c1 to second and c2 to third The NetDeviceContainers created here will be used later, for IP addressing

```
NetDeviceContainer nd0 = csma.Install (c0);  // First LAN
NetDeviceContainer nd1 = csma.Install (c1);  // Second LAN
  NetDeviceContainer nd2 = csma.Install (c2);  // Third LAN
```

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

- Assigning the IP Addresses

```
  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);
  ipv4Addr.SetBase ("10.1.3.0", "255.255.255.0");
  ipv4Addr.Assign (nd2);
```

```
   NS_LOG_INFO ("Configure multicasting.");
```

- *Now we can configure multicasting. As described above, the multicast source is at node zero, for which we assign the IP address as 10.1.1.1 We need to define a multicast group to send packets too. 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:

- Configure a (static) multicast route on node n2 and n1
- Set up a default multicast route on the sender n0
- Have node n6 and n4 join the multicast group We have a helper that can help us with static multicast

```
Ipv4StaticRoutingHelper multicast;
```

- Configure a (static) multicast route on node n1 (multicastRouter)

Node 1 acts as a router

```
Ptr<Node> multicastRouter = c.Get (1);
```

nd0: NetDeviceContainer 0 i.e LAN0, where n1 acts as an input interface for LAN2

```
Ptr<NetDevice> inputIf = nd0.Get (1);
```

A container of output NetDevices (we only need one NetDevice here)

```
NetDeviceContainer outputDevices;
outputDevices.Add (nd2.Get (0));
multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);
```

- Configure a (static) multicast route on node n2 (multicastRouter)

Node 2 acts as a router

```
Ptr<Node> multicastRouter = c.Get (2);
```

nd0: NetDeviceContainer 0 i.e LAN0, where n2 acts as an input interface for LAN1

```
Ptr<NetDevice> inputIf = nd0.Get (2);
```

A container of output NetDevices (we only need one NetDevice here)

```
NetDeviceContainer outputDevices;
outputDevices.Add (nd1.Get (0));
multicast.AddMulticastRoute (multicastRouter, multicastSource,
                             multicastGroup, inputIf, outputDevices);
```

- Set up a default multicast route on the sender n0

Node 0 acts as the sender

```
Ptr<Node> sender = c.Get (0);
```

In LAN0 Node 0 acts as the sender interface to LAN1

```
Ptr<NetDevice> senderIf = nd0.Get (0);
multicast.SetDefaultMulticastRoute (sender, senderIf);
```

- Set up a default multicast route on the sender n0

Node 2 acts as the sender

```
Ptr<Node> sender = c.Get (2);
```

In LAN0 Node 2 acts as the sender interface to LAN1

```
  Ptr<NetDevice> senderIf = nd1.Get (2);
multicast.SetDefaultMulticastRoute (sender, senderIf);
```

(Refer `Scenario 1` for the following code as it uses the same methodology.)

4 Running the scripts in NS3

We usually run the program under the control of Waf. To run a program we simply use –run command along with waf. ie:

```
$ ./waf --run (path)/(program name)
```

Waf checks if the program is built correctly, and then executes it.

4.1 If you don't see the output

If there is some problem running the program, then you may try this command on the CommandLine:

```
$ ./waf configure --build-profile=debug --enable-examples --enable-tests
```

4.2 Debugging

By combining the recipe of the previous one to run the test, we can run the test uner a debugger.

```
$ ./waf --run test-runner --command-template="gdb %s --args
                                              --suite=mytest --verbose"
```

Thank You