Oscar Esparza
Juanjo Alins
Jorge Mata
Jose L. Muñoz

UPC Telematics Department

IP Multicast

0.1 Introduction

Most communications used in the Internet are unicast (one-to-one), meaning that a source sends packets towards the receiver. However, there are other ways of communications: broadcast, multicast, anycast, etc. In particular, in this lab session you are going to deal with group communications, that is to say, multicast communications. According to wikipedia, "multicast (one-to-many or many-to-many distribution) is group communication where information is addressed to a group of destination computers simultaneously." Just emphasize that this lab session is going to deal with IP multicast, that is, multicast at the network layer. However, notice that it is possible to implement multicast at other different layers (for instance at the application layer using other overlays like p2p networks).

The aim of IP multicast is sending IP datagrams to a group of interested receivers in a single transmission. IP multicast tries to use network infrastructure efficiently. To achieve this, the source will send a packet only once, even in case it needs to be delivered to a large number of receivers. Also, routers will replicate this packet to reach multiple receivers, but messages are sent only once over each link of the network. IP multicast is described in RFC 1112, so we recommend that you read this document.

IP multicast is often employed in applications of streaming media, such as Internet television, multipoint videoconferencing, or ghost distribution of backup disk images to multiple computers simultaneously. Obviously, IP multicast can be used both in the Internet and in IP private networks.

There are several problems to be addressed when using IP multicast. In this lab session, we pretend to explain some of the main concepts related to multicast techniques and protocols. In particular:

1. How to identify the receivers that have interest in receiving the multicast flow. We will see that we are going to use class D IP multicast addresses to identify groups of receivers.

2. How the Ethernet layer maps IP multicast addresses into MAC ones. We will see how this direct mapping is constructed.

3. How receivers inform routers about their interest in receiving a multicast flow. We will see that the IGMP protocol is intended to communicate hosts and edge multicast routers.

4. How multicast routers reach all receivers with a single flow per link. We will see that routers use a tree structure to route multicast packets.

Initially, we will address basic aspects of multicast, like obtaining information about existing multicast groups via the DNS, or just sending a ping to a group in the subnet to see how multicast packets are formed. Next, we will discuss more advanced issues, like the sending of a video via multicast in the subnet, the configuration of static multicast routing with the tool `smcroute`, and the transmission of multimedia contents to several remote multicast receivers. Finally, you will transmit a video via streaming, that will be played in the HOST machine. In this lab session you are not going to deal with dynamic multicast routing protocols.

Prior to start this lab session, you should perfectly know about how unicast routing work, and the basics about ICMP and UDP protocols. This lab practice has been designed to last two sessions (4 hours), that should be done in the lab room so the professor can help you to understand the key points. There are also some additional exercises that must be done at home (labeled as "TO DO AT HOME"), just to practice some side aspects about multicasting.

0.2 The scenario of the lab sessions

The aim of this exercise is to get in touch with multicast techniques. The scenario is depicted in Fig 1. As you can see, there are three multicast islands (**SimNet0**, **SimNet3** and **SimNet5**) connected through a public network. There are some routers that are able to manage multicast packets (**R1**, **R2** and **R3**), and a router that is not able to manage this kind of packets (**RC**). These multicast routers are connected via GRE tunnels. There is a GRE tunnel between **R1** and **R2** called `tunnel0`, and another one between **R3** and **R2** called `tunnel1`. This configuration pretends to be similar to the one created some years ago to form the MBONE.

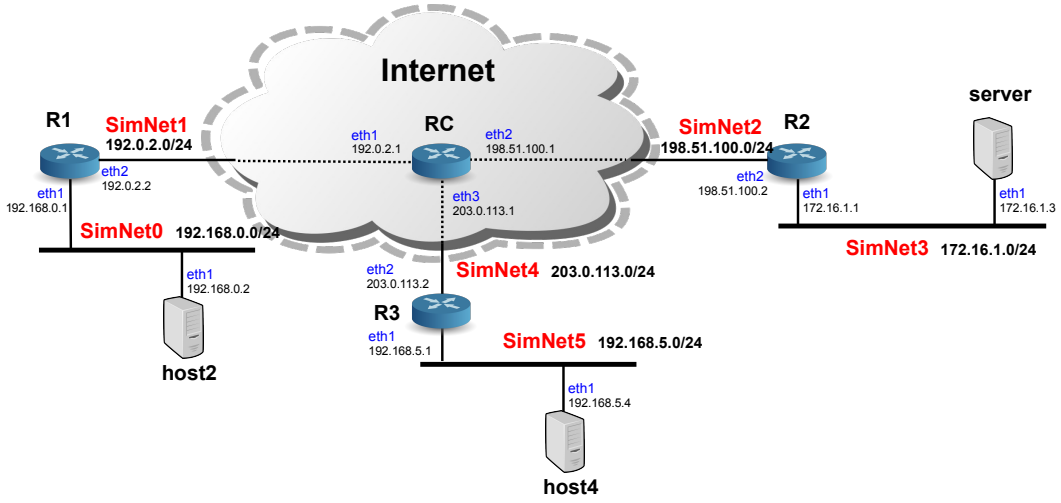

Figure 1: Scenario for the multicast laboratory

0.3 Starting the scenario using simctl

First of all, you will start the virtual scenario with the command:

```
host# simctl ipmulticast start
```

When starting this scenario, IP addresses in network interfaces and routing tables should be properly configured. If not, maybe there is a problem in the scenario or in `simctl`, so contact the professor.

0.4 IP Multicast addresses and DNS

As we mentioned previously, there is a need to identify the receivers that have interest in receiving the multicast flow. When using unicast communications, the `destination address` field of the IP packets identifies an only receiver. However, in multicast communications, this field contains an IP address that identifies a group of receivers that want to receive a multicast flow. For IPv4, the IANA (Internet Assigned Number Authority) reserved the address block `224.0.0.0/4` (class D) for multicast use[1].

The range of addresses `224.0.0.0/24` is also reserved by the IANA for Link-Local multicast addresses, that is to say, addresses reserved to be used in the local subnet. To do so, these packets are sent with `TTL=1`, so routers do not forward them to other networks. For instance, the address `224.0.0.1` identifies `all-hosts`, the address `224.0.0.2` identifies `all routers`. In the slides you have a picture with some of these Link-Local reserved addresses. The range of addresses `224.0.1.0/24` is also reserved by the IANA for some widely-known protocols and applications, but these ones can be forwarded by the routers. For instance, the address `224.0.1.1` is reserved for the Network Time Protocol (NTP). In the slides you also have another picture with some of these reserved addresses. The range of addresses `239.0.0.0/24` is known as Administratively Scoped, meaning that these addresses can be used in private domains (same idea than private unicast addressing).

There are other ranges and addresses reserved for multicast, but we are not going to detail them here. Instead, we are going to obtain information about multicast addresses in an easy and fast way, using the DNS system.

- Execute the following command in your HOST.

[1] IPv6 has also its own reserved block for multicast, in fact, in IPv6 multicast addressing replaces broadcast addressing as it was used in IPv4. Anyway, we are not going to deal with IPv6 in this lab session.

```
HOST$ host all-systems.mcast.net
```

As you can see, the command `host` allows to send requests to the DNS server[2]. In this particular case, we are asking to the DNS the address that corresponds to the name `all-systems.mcast.net`. As you can see, this name corresponds to the address `224.0.0.1`. As we mentioned previously, this particular address is managed by the IANA, and it identifies all (multicast) hosts on the same network segment (TTL=1). Later, you will ping this address. Remember, that there is a list of this reserved names and addresses in the slides, so you can consult them in the DNS using the same command.

- Execute the following command in your HOST:

```
HOST$ host 224.0.0.22
```

As you can see, this address matches the name `igmp.mcast.net`, as it is used to identify routers using the Internet Group Management Protocol (IGMP) in its current version (version 3). Later during the lab session, you will see some IGMP messages that use this address.

0.5 MAC multicast addresses

The next step in this lab session would be to see a multicast packet. Notice that this packet will be just as another IP packet, with the only difference of having as "Destination Address" a multicast one. Hence, this multicast IP packet must also be encapsulated in an Ethernet frame. So, you know the destination IP address, but it is also necessary to put a multicast MAC destination address in the Ethernet frame. Remember that at the beginning of this document we mentioned that this was one of the problems to solve in multicast, and we also mentioned that it was solved by a direct mapping.

Notice that IP addresses has 32 bits, and MAC addresses has 48 bits, so it is possible to directly map an IP address into a MAC address. Notice also that all IP multicast addresses are within this address block `224.0.0.0/4`, so in fact to fully identify a multicast address we only need 28 bits (the 4-bits prefix is fixed, `1110` in binary).

To have a one-to-one mapping (one multicast IP address per one multicast MAC address), we should need multicast MAC addresses to start with a prefix of 20 bits. This means that the IANA should have reserved 16 Organizationally Unique Identifiers (OUI) of 24 bits to the IEEE, but this seemed to be too much money. Instead, there is only one OUI reserved with prefix `01:00:5E` (in hex), and the 25th bit is set to 0. As a conclusion, to construct the multicast MAC address, we take the prefix `01:00:5E` (24 bits), and the 24 remaining bits correspond to `0`+23 least significant bits of the multicast IP address. In the slides there is a figure that explains this. This also means that there is an ambiguity in this process, that is, 32 multicast IP addresses (28-23=5 bits of ambiguity) are mapped in an only MAC address. In this case, the IP layer will solve this ambiguity.

Now you are going to send multicast traffic. Notice that multicast is mainly aimed to send a flow of packets towards a group of receivers, (for instance video streaming). However, just for simplicity, you will start sending a simple ping. So:

- Put a wireshark listening in **SimNet3**.
- Execute the following command in the **server**:

```
server# ping -c 1 224.0.0.1
```

1. How many ICMP packets can you see in **SimNet3**? What type of messages?
2. In the IP header, which is the TTL of this packets? What does this mean?
3. In the IP header, which are the source and destination addresses?

[2]The DNS server can provide you some extra information using the `dig` and `nslookup` commands.

4. In the Ethernet header, which are the source and destination addresses?
5. Can you see the direct mapping in the MAC destination address?

As you can see, the destination MAC address is constructed by using the prefix `01:00:5E` (in hex). The 24 remaining bits correspond to `0`+23 least significant bits of the multicast IP address. In the particular case of the IP address `224.0.0.1`, this corresponds to the MAC address `01:00:5E:00:00:01`.

- Put a wireshark listening in **SimNet3**.
- Execute the following command in the **server**:

```
server# ping -c 1 232.0.0.1
```

1. In this ICMP packet, which is the destination MAC addresses?
2. Can you see any ambiguity in this mapping? Which other addresses cause also this ambiguity?
3. How can the system resolve this ambiguity?

I suppose that you also noticed that there is no reply message for these messages. Just remark that the main purpose of multicasting is not sending pings, and most multicast applications do not require any answer. Anyway, as we want to send multicast pings and receive and answer for them, we have to disable a flag in your Linux system in order not to ignore broadcast and multicast ICMP packets.

- Put a wireshark listening in **SimNet3**.
- Execute the following commands in the **server**:

```
server# echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
server# ping -c3 224.0.0.1
```

1. Is the ping working? Who is answering to the ping?
2. Can you see any reply message in **SimNet3**?

- Now dissable the same flag in **R2**, and ping from **server**:

```
R2# echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
server# ping -c3 224.0.0.1
```

1. Who is now answering to the ping?
2. Can you see any reply message in **SimNet3**?
3. Are the answer packets sent in a multicast way?

0.6 Multicast transmission in a subnet

We pretend to make a multicast transmission of a file between **server** and **R2**, which are located in the same subnet, but before this, let us present you some commands and files that may help you to analyze multicast.

0.6.1 Multicast configuration

There are some files in the system that offers to you some information about multicast interfaces and groups. Have a look to the following configuration commands and files[3]:

- `/proc/net/igmp`: this file contains the groups to which the host has joined. If you have a look to this file in **server**, you will see something like this:

```
server:~# cat /proc/net/igmp
Idx     Device    : Count Querier       Group    Users Timer    Reporter
1       lo        :     1      V3
                              010000E0     1 0:00000000               0
3       eth1      :     1      V3
                              010000E0     1 0:00000000               0
```

  As you can see, this file shows you information about the interfaces that are joined to certain groups (the loopback interface `lo` and `eth1` are joined to the group `010000E0`. This notation seems odd, but it is quite typical. If you translate this hex string `010000E0` to decimal you will see that is equal to `1.0.0.224`. Does it sound familiar to you? This way provides another way of denoting IP addresses, just changing the order of the octets. You can also see this same information by executing the command `netstat -gn`.

- `/proc/net/ip_mr_vif` : this file contains the interfaces that are involved in multicast operations in the multicast router, and some statistics of usage (e.g. number of packets sent and received). If you see the contents of this file right now you will see the file is empty. This is totally normal, as multicast information can only be seen meanwhile there is a multicast flow being transmitted.

- `/proc/net/ip_mr_cache` : this file shows the contents of the Multicast Forwarding Cache. As it happens in the previous file, if you have a look to this file right now, you will see nothing, as the file only shows active routes. Anyway, the way in which this file displays information is sometimes hard to read, so we recommend to use the command `ip mroute show`, which shows just the same information.

0.6.2 Transport protocol

ICMP is a layer-3 protocol whose aim is mainly reporting errors of the network layer, so it is not suitable as transport protocol to send data. So, which is the transport protocol to send information when using multicast? TCP does not support multicast, for several reasons. For instance, multicast is not a connection-oriented mechanism. Also, multicast is often used for applications such as audio and video streaming, where an occasional dropped packet is not a problem, so reliability offered by TCP may not be appropriate. For this reason, most of the times IP multicast uses UDP as transport layer protocol, and you are going to use it here in this lab session.

As you should know, UDP is an unreliable protocol by nature, meaning that UDP packets can be lost, duplicated or delivered out of order. In case of requiring reliability to distribute critical data during the transmission of multicast packets, it should be provided by the application layer, or by using other transport protocols, for instance the Pragmatic General Multicast (PGM) protocol (RFC 3208), which have been developed to detect losses and retransmit packets. In this lab session, we are not going to deal with reliable multicast

0.6.3 `udp-sender`

You are going to use `udp-sender` and `upd-receiver` to make the transmission of a video file via multicast. Prior to start this transmission, let us test the behavior of these commands. We will start with `udp-sender`, here you have the available options, extracted from the man page.

[3]Remember that to see the contents of a text file you have to use the command `cat`

```
udp-sender [--file file] [--full-duplex] [--half-duplex] [--pipe pipe]
[--portbase portbase] [--blocksize size] [--interface net-interface]
[--mcast-addr data-mcast-address] [--mcast-all-addr mcast-all-address]
[--max-bitrate bitrate] [--pointopoint] [--async] [--log file]
[--min-slice-size min] [--max-slice-size max] [--slice-size] [--ttl
time-to-live] [--fec stripesxredundancy/stripesize] [--print-seed]
[--rexmit-hello-interval interval] [--autostart autostart] [--broad-
cast] [--min-clients clients] [--min-wait sec] [--max-wait sec]
[--nokbd] [--retriesUntilDrop n]
```

As you can see, `udp-sender` offers several options to send information via broadcast or multicast. It can be used in a unidirectional or bidirectional way, and it is possible to tune many application and network parameters, like the TTL, the maximum bitrate, retransmissions, FEC (Forward Error Correction) codes, etc. Do not worry because in the lab session we are not going to use all these options, but it would be a good idea to have a look to them so you can understand their meaning. You must know that this software uses two multicast groups to enable the transmission:

- A multicast group to send data: in our case you will use the multicast group `232.43.211.234`.
- A multicast group to control and add reliability, and you will use `225.1.2.3`.

So let us start testing `udp-sender`:

- Open another console of **server**, that will be used to see the status of this host meanwhile you are sending the video. Go to the **HOST** machine and execute:

```
HOST$ simctl ipmulticast get server 1
```

  This command will open console 1 of **server** (you where previously working using console 0). Now, in this console 1, see the contents of the file `/proc/net/igmp`:

```
server# cat /proc/net/igmp
```

- Put a wireshark listening in **SimNet3**.
- In this case, **server** will act as video server. So, go to console 0 and execute the following command:

```
server# udp-sender --file=./big_664.mpg --min-clients 1 --portbase 22345 \
--nopointopoint --interface eth1 --ttl 1 --mcast-addr 232.43.211.234 \
--mcast-all-addr 225.1.2.3
```

- The previous command prepares **server** to transmit via multicast the locally stored video file `big_664.mpg`. Have a look to the manual of this command (`man udp-sender`) and explain each of the options used.
- In **SimNet3**:

  1. How many packets can you see? Which type of packets?
  2. In the UDP packet, which is the destination address? and the TTL?
  3. Which are the port numbers?
  4. Can you see any IGMP packet? Which type of packet? Which is the IP destination address of this IGMP messages?

As you can see, there is an UDP packet sent towards the control multicast group, whose objective is contacting with the destination. As the client has not been launched, the server just waits to send the video. Notice that the TTL value of this IP packet is 1, as specified by the application. In this particular case, the `-ttl 1` option is optional, because applications that do not specify the TTL should put 1 as default value for multicast packets (have a look to RFC 1112). However, remember to put the proper TTL when trying to reach remote destinations.

There are also a couple of IGMP messages, in which the **server** tries to join the control multicast group. As there is no multicast router, there is no answer to these messages.

- Now have a look to the console 1 of **server**, and again see the contents of the file `/proc/net/igmp`.

  1. Can you see any difference?
  2. What is the hex and the dot-decimal notation of the newly added multicast group?

- Go to console 0, and close the server by typing `Control+C`.

  1. Can you see in `SimNet3` any new IGMP message? Which type?
  2. After closing, can you see again any change in the `/proc/net/igmp` file?

  As you can see, when closing `udp-sender` the **sender** leaves the control group by sending two IGMP Leave Messages. No answer will be sent to these messages as there is no multicast router.

0.6.4 upd-receiver

The command `udp-sender` works with its corresponding client, `udp-receiver`, here you have the available options, extracted from the man page:

```
udp-receiver [--file file] [--pipe pipe] [--portbase portbase]
[--interface net-interface] [--log file] [--ttl time-to-live]
[--mcast-all-addr mcast-all-address] [--nokbd] [--exitWait millisec-
onds]
```

Let us start testing `udp-receiver`:

- Open another console of **R2**, that will be used to see the status of this host meanwhile you are launching `udp-receiver`. Go to the **HOST** machine and execute:

```
HOST$ simctl ipmulticast get R2 1
```

This command will open console 1 of **R2** (you where previously working using console 0). Now, in this console 1, see the contents of the file `/proc/net/igmp`:

```
server# cat /proc/net/igmp
```

1. Which interfaces in **R2** are joined to group `224.0.0.1`?

- Now open two wiresharks, one listening in **SimNet2** and another one in **SimNet3**.
- In this case, **R2** will act as a host, receiving the video. So, go to console 0 and execute the following command:

```
R2# udp-receiver --file=big_664.mpg --mcast-all-addr 225.1.2.3 --ttl 1 \
--portbase 22345
```

This command prepares **R2** to receive a video via multicast, which should be stored locally with name `big_664.mpg`.

- Have a look to the wiresharks.

  1. How many packets can you see in **SimNet3**? and in **SimNet2**?
  2. Why do you think that these packets are in this network and not in the other?

  According to RFC 1112, applications sending multicast packets should choose which are the output interfaces of these packets, and if not, the operating system will choose them. The application `udp-receiver` has an option to select the interface to send packets, but you did not use it, so the application has chosen the interface that corresponds to the default unicast route.

  Later you will change this to make multicast packets to go through **SimNet3**, but now let us have a look to packets in **SimNet2** using wireshark.

  1. In the UDP packets, which is the destination address? and the TTL?
  2. Which are the port numbers? Are these numbers familiar to you?
  3. Can you see any IGMP packet?

  As you can see, there is a couple of UDP packets sent towards the control multicast group, whose objective is contacting with the server. As the server has not been launched, nothing happens. There are also a couple of IGMP Membership Report messages, in which **R2** tries to join the control multicast group. As there is no multicast router, there is no answer to these messages.

- Now have a look to the console 1 of **R2**, and again see the contents of the file `/proc/net/igmp`.

  1. Can you see any difference?

- Go to console 0, and close `udp-receiver` by typing `Control+C`.

  1. Can you see in `SimNet2` any new IGMP message? Which type?
  2. After closing, can you see again any change in the `/proc/net/igmp` file?

  As you can see, when closing `udp-receiver` **R2** leaves the control group by sending two IGMP Leave Messages. No answer will be sent to these messages as there is no multicast router.

0.6.5 Sending the video file in the subnet

Now that you know how `udp-sender` and `udp-receiver` behave, it is time to make a real distribution of a video file via multicast:

- Put a wireshark listening in **SimNet3**.
- Go to console 0 of **server** and execute again the following command:

```
server# udp-sender --file=./big_664.mpg --min-clients 1 --portbase 22345 \
--nopointopoint --interface eth1 --ttl 1 --mcast-addr 232.43.211.234 \
--mcast-all-addr 225.1.2.3
```

- Go to console 0 of **R2** and execute the following command, now forcing the packets to be sent to the proper interface:

```
R2# udp-receiver --file=big_664.mpg --mcast-all-addr 225.1.2.3 --ttl 1 \
--interface eth1 --portbase 22345
```

  1. Was the transmission ok?
  2. How many packets can you see in wireshark?

As you have seen, the transmission went ok, but now it is time to focus on some key point in this transmission.

- Let us start analyzing the four initial UDP packets of short-length.

  1. What do you think is the purpose of these initial packets?
  2. In the first UDP packet sent by **server**, which is the IP destination address?
  3. Do you recognize anything inside of the DATA field of this packet? Try to convert `232.43.211.234` in hex.
  4. At the time this first packet was sent by the **server**, was **R2** listening at the control multicast address?
  5. What do you think is the purpose of the second and third UDP packets (the ones sent by **R2**)?
  6. At this time, does **R2** know which is the data multicast group that **server** is going to use to send the video?
  7. Have a look to the fourth UDP packet. Is it a multicast or unicast packet? What is the purpose of this message? Do you recognize anything in the data field of the packet?

  As you can see, this first dialogue of four short-length UDP packets is used for sender and receiver to mutually recognize themselves and exchange key information about the transmission. The sender knows which is the data multicast group, but not the receiver. Until the last fourth packet the receiver is not informed (in a unicast way) about the data multicast group that has to join. For this reason, the following message is an IGMP Membership Report of **R2** joining both groups. You can verify that **R2** is joined to both groups by having a look to file `/proc/net/igmp` meanwhile the file is downloading.

- Regarding the transmission of the video (look at packets that are full of information):

  1. Which is the destination IP address of these packets?
  2. What is the size of most of this data packets?
  3. There are some unicast packets from **R2** to **server**. What do you think is the purpose of these feedback packets?

  As you have seen, **server** sends the video by means of several UDP packets (full of data) with destination address `232.43.211.234`. There are also some feedback unicast packets from **R2** to **server**, used for reliability purposes[4]. Finally, both sender and receiver leave the corresponding multicast groups by sending IGMP Leave Messages.

Now you know how to transmit information via multicast in a subnet. However, you know that most remote locations are far away, so routers should be used to reach them. It is time you go a step further, trying to do the same thing, but now involving multicast routers during the transmission.

0.7 Multicast and routing

In previous sections you only used the virtual machines in **SimNet3**, but now you will use the rest of the scenario. Remember that in the scenario of Fig 1 you have three multicast islands (**SimNet0**, **SimNet3** and **SimNet5**) connected through a public network. There are some multicast routers (**R1**, **R2** and **R3**), and a router without multicast capacities (**RC**). We mentioned that multicast islands were connected via GRE tunnels, but in fact you are going to create them right now. There should be a GRE tunnel between **R1** and **R2** called `tunnel0`, and another one between **R3** and **R2** called `tunnel1`.

[4]These feedback packets may be avoided in case of having a unidirectional channel.

0.7.1 Configuring GRE tunnels

In a previous lab session you learned how to configure and test IPIP tunnels. Unfortunately, Linux does not directly support multicast transmission via IPIP tunnels, and for this reason we are going to use GRE tunnels. If you want to know more about GRE tunnels, you can read RFC 1701 and RFC 1702.

You are not going to configure these tunnels manually, but using a script. Anyway, here you have a summary of the command that should be executed in the encapsulator and decapsulator to do so (you DO NOT have to execute them):

- Configure the tunnel interface by means of the command `ip tunnel`. Tunnels have to work in **gre** mode. As an example, in **R2** the command to execute:

```
R2# ip tunnel add tunnel0 mode gre local 198.51.100.2 remote 192.0.2.2 dev eth2
```

- Activate the new virtual interface of type tunnel. To do so, it is necessary to assign an IP address to it (for instance, by means of the `ifconfig` command). In Linux, it is necessary to assign an IP address to any interface if we want it to properly work. Just as an example:

```
R2# ifconfig tunnel0 192.168.110.1
```

- Update the routing table to make the proper packets to be routed towards the tunnel (for instance, by means of the `route` command). Just as an example:

```
R2# route add -net 192.168.0.0/24 dev tunnel0
```

To create both tunnels automatically, go to the console of your HOST machine, and execute:

```
HOST$ simctl ipmulticast exec addtun
```

After creating the tunnels, you have to verify that they properly work.

- Put a wireshark listening to interface **SimNet2**.
- Send a ping from the **server** to **host2**.
  1. Is the ping working?
  2. Can you see packets in the wireshark? What type of packets? Describe the headers you see.
  3. Regarding the IP outer header, which are the source and destination addresses?
  4. Which is the size of the GRE header?
  5. Which is the source and destination addresses in the inner header?
- Send also a ping from the **server** to **host4** to verify that `tunnel1` works.

If you detect any kind or error about any of both tunnels `tunnel0` and `tunnel1`, contact the professor to verify if there is any error in the machine or in `simctl`.

0.7.2 Multicast and ICMP

Just in previous sections you used pings to test the tunnels and to send multicast packets in the subnet. In both cases, pings were used just for easiness, and because you are very used to send and analyze "ICMP Echo Request" and "ICMP Echo Reply" messages using `wireshark`. Anyway, remember that ICMP is not only to send pings, it is the protocol intended to report the errors of the network. However, you must know that ICMP is barely used when sending multicast traffic. As clearly stated in RFC 1112 " Host Extensions for IP Multicasting":

```
An ICMP error message (Destination Unreachable, Time Exceeded,
Parameter Problem, Source Quench, or Redirect) is
never generated in response to a datagram destined to an IP host group.
```

These error messages are in fact the most typical ones, so this means that ICMP is not going to help you during the sending process of multicast packets, sending errors in case you commit a mistake. You are not going to receive any message like "Host Unreachable", "Fragmentation Needed", or "Time Exceeded" due to an error generated by a multicast packet. Next, you are going to deal with multicast and IP tunneling, so the scenario is prone to errors generated by TTL and MTU problems (as you noticed in the IP tunneling lab session). So, take this into account when configuring the network and when sending multicast packets. You should pay special attention when defining the TTL in each multicast packet (the scope of the packet), and also the size of them so they can traverse tunnels properly.

0.7.3 Multicast static routing: `smcroute`

So you are ready to create the multicast routing tree from the sender to all the receivers. You are not going to use dynamical multicast routing, but static one by means of the `smcroute` command. This `smcroute` command is able to manipulate the multicast static routes of the Linux kernel. We recommend to you to consult the manual of `smcroute` to become familiar with it.

You will start configuring the multicast static routing tree to communicate the two multicast islands **SimNet0** and **SimNet3**, which remember that are connected via the gre tunnel `tunnel0`. Later you will connect the other island of **SimNet5**. Notice that this configuration pretends to be similar to the one created some years ago in the MBONE.

- Have a look to the file `/proc/net/ip_mr_vif` in router **R2**. This file should be empty because the router has not yet taken any action.
- Open four wiresharks, listening from **SimNet0** to **SimNet3**.
- Start the daemon `smcroute` in router **R2**.

```
R2# smcroute -d
```

  See again the contents of `/proc/net/ip_mr_vif`. What changes do you see?

  This file `/proc/net/ip_mr_vif` will show you which are the interfaces which are able to use multicast in the multicast router, and some statistics of their use. As you have not sent any multicast packet, you will see that the statistics are empty.

- Execute the following command in **R2** to add a new multicast static route:

```
R2# smcroute -a eth1 0.0.0.0 232.43.211.234 tunnel0
```

  This command will route IP datagrams entering through `eth1` with any origin address and destination address the multicast group `232.43.211.234` towards the tunnel (so they will reach the other island).

- Execute the following command to make **R2** to join the entry interface `eth1` to the multicast group `232.43.211.234`:

```
R2# smcroute -j eth1 232.43.211.234
```

  1. View the content of the file `/proc/net/igmp`. What changes do you see in this file?
  2. Have a look to **SimNet3**. Can you see any IGMP message in that interface?

- Now send a ping from **server** to the multicast group `232.43.211.234` with a scope of three hops.

```
server# ping -t3 -c1 232.43.211.234
```

1. Is the ping working? In which SimNets can you see ICMP packets?
2. Can you see any encapsulation in the ICMP packet in **SimNet1** and **SimNet2**? What is the size of a GRE header?
3. Have a look again to file `/proc/net/ip_mr_vif`. Do you see any change in the statistics related to `tunnel0`?
4. Execute `ip mroute show` to see the Multicast Forwarding Cache. What is the meaning of all these parameters?
5. Why do you think **R1** is not forwarding the packet?

According to wireshark, you will see that the packet has been introduced in the tunnel, but the decapsulator **R1** has not been configured as multicast router, so the ICMP packet is not forwarded to **SimNet0**. Notice also that you have configured in **R2** a unidirectional static multicast route, that is, from the server in **SimNet3** to possible receivers in **SimNet0**.

- Using `smcroute`, configure **R1** to route IP datagrams entering through the tunnel with any origin address and destination address the multicast group `232.43.211.234` towards `eth1`. To do so, just follow the same procedure that you used in **R2**.

1. Can you see the packets in **SimNet0**? Which type of packets?
2. Can you see any ICMP Echo Reply? Why?
3. Which is the TTL value of this packet in **SimNet0**?
4. If you send the same ping from **server** but with TTL=2, would it work?

If you made the previous configuration properly, when you ping the multicast group from **server** with TTL=3, you will see the packet to reach **SimNet0**. If this is not happening, review the configuration because there is something wrong. There will be no ICMP Echo Reply message, as there is not an application running there ready to answer. Notice that it is important to define the proper scope for multicast packets. With TTL=2 you will not reach the destination. With TTL=4 or higher you will reach destination, but maybe you will also send packets to other networks that you did not pretend to reach, wasting resources (bandwidth, CPU, etc.). So be careful when defining the scope of your multicast packets.

0.7.4 Testing tools

You used `smcroute` to create a quite simplistic static multicast tree to connect **SimNet3** to **SimNet0**. Once configured, you need ways to test if it works well. You previously used ping to do so, but ping is not an application that send real data. Also, there are some other problems that appear when using ICMP packets to test networks, just remember that some network administrators filter them just to avoid security attacks (you made some tests about this matter in the IP tunnel lab session).

So, here you will use two tools to test that multicast is working in the right manner.

`ssmping`

You are going to use the application `ssmping`, which allows you to verify both unicast and multicast routing. Have a look to the on-line manual of this program to become familiar with it. Basically, `ssmping` allows to verify if a host can receive SSM (Source Specific Multicast) packets from another host, by sending unicast and multicast UDP packets (remember that TCP does not support multicast traffic).

- Put wiresharks listening in all the involved networks.
- In console 0 of **server** start the daemon of the application executing the command

```
server# ssmpingd
```

When you consider necessary, use console 1 to control if **server** has joined to any new multicast group (use `netstat -gn`).

- In console 0 **host2**, execute the command:

```
host2# ssmping 172.16.1.3
```

When you consider necessary, use console 1 of **host2** to control the same thing.

1. What output can you see in **host2**? and in **server**?
2. According to this output, do you consider that the multicast routing is working?

Notice that `ssmping` is able to test both unicast and multicast routing. Have a look to the UDP packets. Notice that the behavior is periodical, and groups of three UDP packets are sent every second.

1. What is the destination and origin address of the first packet? Is a unicast or multicast packet?
2. And the second one? What is the purpose of these two packets?
3. Is the third packet different? Who is the origin and destination? Unicast or multicast? So, what is this packet testing? Unidirectional or bidirectional?

Regarding IGMP packets:

1. In **SimNet0**, can you see any IGMP Message? What kind of message? Which is the multicast group involved? (for easiness, put `igmp` in the *Filter* option available in the top-left corner of wireshark).
2. Is this information coherent with what you see using `netstat -gn`?
3. Is there any other IGMP message in the rest of networks?

- Now that you know what is the purpose of `ssmping`, and how it works, just test the same in the inverse direction (execute `ssmping` in **server** and `ssmpingd` in **host2**). Prior to execute the commands, try to foresee what is going to happen. Analyze the wireshark traces and extract your own conclusions.

`mcsender` and `emcast` (TO DO AT HOME)

It is also possible to verify the correct use of multicast static routing using the applications `mcsender` and `emcast`. Have a look to the manual of both applications to see how they work.

- Put wiresharks listening in all the involved networks.
- In **server** (acting as issuer) execute `mcsender` to send packets to the multicast group 232.43.211.234 using the UDP port 12345 with a scope of three hops.
- In **host2** (acting as receiver) execute `emcast` to join group 232.43.211.234 using UDP port 12345.

1. What output can you see in **host2**? and in **server**?
2. According to this output, do you consider that the multicast routing is working?
3. What is the destination and origin address of the packets? Are they unicast or multicast?
4. Can you see any IGMP Message? Where? What kind of message? Which is the multicast group involved?

0.7.5 Trees with multiple branches

So now you know how to create the multicast static routing tree to communicate the two multicast islands **SimNet0** and **SimNet3**. But, what happens with the other island **SimNet5**? What should you do to create a tree with multiple branches? Your objective now is to create the multicast static routing tree necessary to perform a multicast transmission from **server** to receivers located at both **SimNet0** and **SimNet5**.

The command `smcroute` allows us to create this configuration in an easy way, by only changing the configuration in **R2**.

- Put wiresharks listening in all the involved networks.
- Kill the `smcroute` daemon, so all previous configuration in that router will be lost

```
R2# smcroute -k
```

- Start again the daemon `smcroute`.

```
R2# smcroute -d
```

- Here you will see the main difference regarding the previous configuration. When adding the new multicast static route, instead of an only output interface, you can put a list of interfaces that will be used as output. So, execute the following command:

```
R2# smcroute -a eth1 0.0.0.0 232.43.211.234 tunnel0 tunnel1
```

- Join the entry interface `eth1` to the multicast group `232.43.211.234`:

```
R2# smcroute -j eth1 232.43.211.234
```

- Use `ssmping` to test the configuration, executing these commands:

```
server# ssmpingd
host2# ssmping 172.16.1.3
host4# ssmping 172.16.1.3
```

  1. Is `ssmping` working? What output message can you see in **host2**? **host4**? And in **server**?
  2. What will deduce from this output messages? What do you consider is missing?
  3. Have a look to **SimNet4**, what type of messages can you see? unicast or multicast? And in **SimNet5**?

- Make the necessary changes to make the multicast routing work so you can send a multicast flow from **SimNet0** to both **SimNet3** and **SimNet5**.
- Test that the configuration is properly working, with `ssmping`.
- Use also `mcsender` and `emcast` to test the configuration.

0.7.6 Sending the video

Now you are ready to make a real distribution of a video via multicast, but now between remote multicast islands. For the video distribution, you will use `udp-sender` and `udp-receiver` again. Remember that this software uses two multicast groups to enable the transmission: a multicast group to send data and a multicast group to control and add reliability to the transmission. In our case, you will use the multicast group 232.43.211.234 (previously configured in our scenario) to send data. As control group you will use 225.1.2.3.

- Configure the multicast routers for the control group 225.1.2.3, exactly in the same way you did previously with the data group 232.43.211.234.

- Test that this control group 225.1.2.3 is properly transmitted by the multicast tree from **SimNet0** to both **SimNet3** and **SimNet5**.

- Start `udp-receiver` in both **host2** and **host4**:

```
host2# udp-receiver --file=big_664_in_host2.mpg --mcast-all-addr 225.1.2.3  \
--ttl 3 --interface eth1 --portbase 22345
host4# udp-receiver --file=big_664_in_host4.mpg --mcast-all-addr 225.1.2.3  \
--ttl 3 --interface eth1 --portbase 22345
```

- Start `upd-sender` to transmit the video `big_664.mpg` from **server**. Remember that your multicast routing tree is unidirectional, and hence you will not receive any feedback from the receivers (for that reason we use the options `autostart` and `max-bitrate`. Notice also that you need to put the proper `ttl` and `blocksize` values to properly traverse the tunnels. Execute:

```
server# udp-sender --file=./big_664.mpg --portbase 22345 --nopointopoint \
--interface eth1 --ttl 3 --mcast-addr 232.43.211.234 --mcast-all-addr 225.1.2.3 \
--nokbd --async --max-bitrate 900K --autostart 1 --blocksize 1400
```

  If the network is properly configured, you will see that the file is being transmitted via multicast and it is captured by both receivers. Once finished (after 2-3 minutes), a copy of the file should be stored in both **host2** and **host4**. If you want to verify that these copies are equal to the one stored in **server**, you can use `md5sum`, which computes a MD5 hash of the file. The result of this `md5sum` must be the same in all the hosts.

```
server~# md5sum big_664.mpg
cf88e5d297fc7458e3914c224af4c06f  big_664.mpg
```

  Remember that the value may be different as we did not implement any reliability mechanism (there is no feedback to ask for retransmissions, and you do not use any FEC technique to correct errors).

0.7.7 Bidirectional tree (TO DO AT HOME)

Creating a bidirectional tree is mandatory to offer reliability to the multicast transmission. As we mentioned, we are not going to deal with reliable multicast, but you should be able to create this bidirectional tree using `smcroute`.

- Configure the multicast routers to create this bidirectional tree for both data and control groups 232.43.211.234 and 225.1.2.3. Notice that in the previous section you created the forward tree (from **server** to **host2** and **host4**). Now you only need to create the backward tree (from **host2** and **host4** to the **server**).

- Start `udp-receiver` in both **host2** and **host4**:

```
host2# udp-receiver --file=big_664_in_host2.mpg --mcast-all-addr 225.1.2.3 \
 --ttl 3 --interface eth1 --portbase 22345
host4# udp-receiver --file=big_664_in_host4.mpg --mcast-all-addr 225.1.2.3 \
--ttl 3 --interface eth1 --portbase 22345
```

- Start `upd-sender` to transmit the video `big_664.mpg` from **server**. Now the multicast routing tree is bidirectional, so we can receive feedback from receivers, and the options `autostart` and `max-bitrate` can be avoided as it is possible to establish a certain flow control between sender and receiver. Execute:

```
server# udp-sender --file=./big_664.mpg --portbase 22345 --nopointopoint \
--interface eth1 --ttl 3 --mcast-addr 232.43.211.234 --mcast-all-addr 225.1.2.3 \
--nokbd --autostart 1 --blocksize 1400
```

As there is no limitation in the bitrate, you will notice that the transmission is much more faster than in the previous case.

- Using wireshark, have a look to the UDP packets sent towards the data multicast group.

  1. Are these UDP packets of maximum size?
  2. Do you know all the headers involved? Is there any header related to the application `udp-sender`?
  3. Which is the maximum value of the `-blocksize` option of `udp-sender` in order to traverse the tunnel with no problems?

  1. Which percentage of packets are feedback packets?
  2. Are these feedback packets unicast or multicast?

0.7.8 Live streaming (TO DO AT HOME)

Until now you just sent files over the network, but notice that there are lots of applications using multicast that send information via streaming. So, you are going to send the same video via multicast from **server** to both **host2** and **host4**, but at the same time your HOST machine will act as receiver playing the video in streaming. Prior to do this, you have to prepare your HOST machine to interact with the virtualized scenario. Next, we are going to provide some guidelines to play this video in streaming, but this should be considered as an example, and you may need to adapt the commands to the particular configuration of your HOST machine.

- First thing to do, you will need to install in your HOST machine `udpcast`[5], which includes `udp-sender` and `udp-receiver`. Notice that the version of `udpcast` that you are installing in your HOST machine may be different to the one installed in the virtual machines.
- You will also need a video player that allows to play in streaming, for instance `vlc`[6].
- Next, you have to configure the interface you are going to use in your HOST machine to interact with the virtualized scenario. Using `ifconfig`, assign the IP address 172.16.1.4 to **SimNet3**.
- Using `route`, create a new route to send the range 224.0.0.0/4 to **SimNet3** (if not, multicast packets will be sent towards the default route).
- Execute `udp-receiver` in **host2** and **host4**.

```
host2# udp-receiver --file=big_664_in_host2.mpg --mcast-all-addr 225.1.2.3 \
--ttl 3 --interface eth1 --portbase 22345
host4# udp-receiver --file=big_664_in_host4.mpg --mcast-all-addr 225.1.2.3 \
--ttl 3 --interface eth1 --portbase 22345
```

- Execute `udp-receiver` in your **HOST** machine, but redirecting the output of this application to the input video player `vlc`. As we mentioned, maybe you will need to adapt the command to the particular characteristics of your HOST machine. So, just as an example, execute:

```
HOST$ udp-receiver --nokbd --mcast-rdv-address 225.1.2.3 --ttl 1 \
--interface SimNet3 --portbase 22345 | vlc --file-caching 0 - 2>/dev/null
```

[5] Use `sudo apt-get install udpcast`
[6] Use `sudo apt-get install vlc-nox`

Notice that the option `-mcast-rdv-address` of this `upd-receiver` is different from the one used previously (`-mcast-all-addr`), because the versions of `udpcast` are different.

- Finally, execute `udp-sender` at **server**:

```
server# udp-sender --file=./big_664.mpg --portbase 22345 --nopointopoint \
--interface eth1 --ttl 3 --mcast-addr 232.43.211.234 --mcast-all-addr 225.1.2.3 \
--nokbd --autostart 1 --blocksize 1400
```

If everything is properly configured, the file will be transmitted via multicast from **server** to the three receivers: **host2** and **host4** receiving the file, and **HOST** playing in streaming the video.

0.7.9 The end of the overlay (TO DO AT HOME)

We mentioned that the configuration of this scenario pretends to be similar to the one used for the gradual deployment of the multicast technology (MBONE). In fact, there are other examples in which an overlay was constructed to help during the deployment of a new technology, connecting islands by using tunneling and encapsulation (for instance, the slow deployment of the IPv6 protocol). As time goes by, there are more nodes into the overlay (because they are aware of the new technology, for instance multicast), and few out of it. Finally, it is expected that all nodes are able to understand the new technology, so the overlay may have no sense (nor the tunnels).

To try to test this, let us try to translate this same idea to our multicast scenario. At the beginning we mentioned that **RC** was not a multicast router, so it was not possible for this router to be part of the multicast tree. Instead, you used IP tunnels to construct an overlay of multicast routers, bypassing **RC** by means of the GRE tunnels. This overlay works well, but you know that it is not totally transparent, and in fact there are some performance problems. For instance, every time you send a message through the multicast tree, this message is sent twice in **SimNet2** (one through `tunnel0` and another one through `tunnel1`), wasting bandwidth. You should have noticed this, this design is not able to use the network infrastructure efficiently and to achieve one of the main requirements of multicast: "messages must be sent only once over each link of the network". Anyway, the overlay works pretty well, and this is just one more of the side effects caused by the use of tunneling techniques (overhead caused by the tunnel headers, MTU problems, TTL, etc.).

Now consider that **RC** evolves technologically, and now it is able to be a multicast router. So, it should be included as part of the multicast tree. In fact, now all routers are multicast routers, and the overlay and the tunnels do not have any sense.

So, as a final exercise for this lab session, include **RC** in the multicast tree and send again the video from **server** to the two receivers: **host2** and **host4**

- Delete the static multicast tree by killing smcroute in all the multicast routers. Instead of doing it manually, execute the following command in your HOST:

```
HOST$ simctl ipmulticast exec mcclean
```

- Delete the tunnels by executing the following command in your HOST:

```
HOST$ simctl ipmulticast exec deltun
```

  Notice that deleting the tunnels means that you cannot access to the private networks (using unicast).

- Use `smcroute` to create a tree from **SimNet3** to **SimNet0** and **SimNet5**. Notice that the tunnels do not exist, and **RC** should be included in the multicast tree.

  1. How many commands did you execute in every router?
  2. Do you think that static multicast routing is worthy? What would happen in case of having much more routers to configure?

- Use `udp-sender` and `udp-receiver` to make the video transmission.

    1. How many times the same flow is sent over each link?
    2. What is now the maximum block size allowed in `udp-sender`?
    3. And the TTL?
    4. Can you use a bidireccional transmission (with feedback information)?
    5. In terms of performance, do you think that this solution works better than the overlay?

As you can see, you have executed several commands to make this configuration possible. Multicast static routing is easy to configure and to understand in case of having small networks, but obviously is not an scalable option in case of having mid to large network topologies. In that case, dynamical multicast routing protocols like Protocol Independent Multicast (PIM) should be used. We are not going to deal with dynamical multicast routing in this lab session. We hope that all these experiments helped you to understand how complicated is to understand multicast, even in this simplistic scenario with only some few routers.