H2020 iP Over IcN- the betTer IP (POINT)

Examples

Deploy an IP Multicast scenario

Authors: Yannis Thomas, Alexander Phinikarides

1. Overview

This document shows how to setup an IP Multicast scenario, using the IP Multicast / IGMP handler of POINT. Two different examples are provided.

1. The baseline example uses a few VMs directly connected via Ethernet, with dummy applications to generate and consume IP multicast traffic.
2. The advanced example with mininet based devices.

The examples were tested with Debian 8 and 9 machines using kernel 3.16 and 4.10, respectively, running on either KVM or VirtualBox.

2. Baseline Example

The purpose of this example is to document the creation of simple IP multicast network, using directly connected VMs and simplistic client and server applications, that do not exhibit the full functionality of an actual IP multicast setup.

2.1 Topology

Figure 1 depicts the baseline testbed topology used as a reference setup in the next sections. It introduces two client NAPs (cNAPs), a single server NAP (sNAP) and one IP end-host per IP network; however, multiple users per NAP are also supported.

The figure also depicts important details, such as IP addresses, NIC names, link types etc, for deploying the entire testbed on a single physical node. In this case, each node represents a VM with the illustrated links, network interfaces and IP addresses. Note also that the testbed can be created with just 4 VMs, when the IP users are collocated with the NAPs.

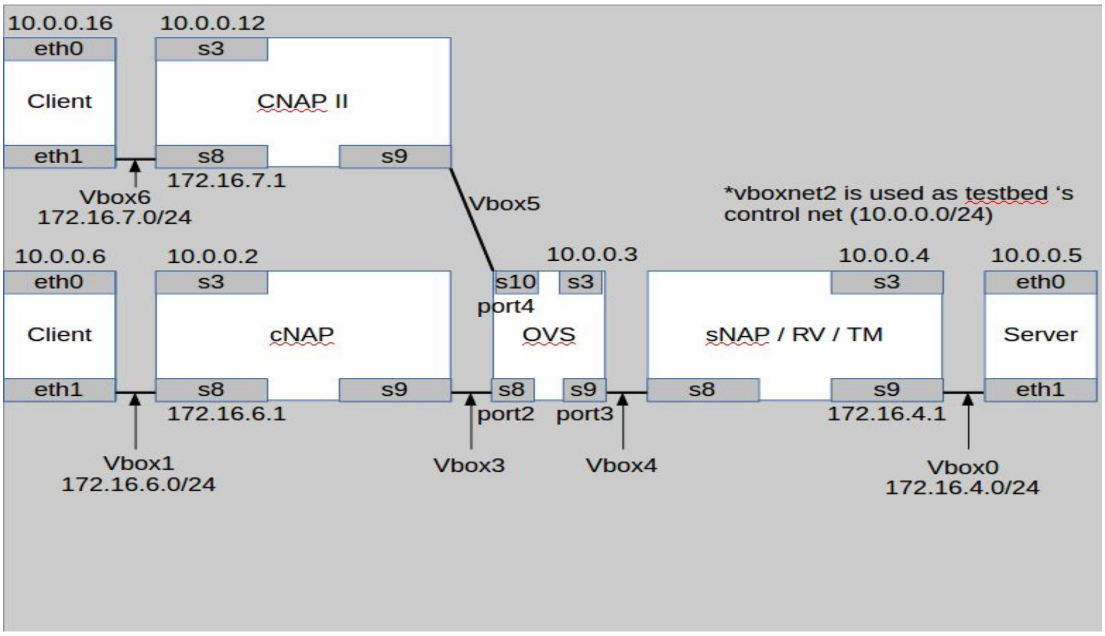

Figure 1: Baseline topology.

The IP network address ranges used are:
10.0.0.0/8 – control network
172.16.4.0/8 – sNAP IP network
172.16.6.0/8 – cNAP I IP network
172.16.16.0/8 – cNAP II IP network

2.2 Node preparation

IGMP traffic must be pushed to the appropriate NAP (interface), based on the used destination IP address. The related addresses are *225.0.0.0/32* and *224.0.0.0/32*, so each client should route these packets to the appropriate IP interface of the NAP.

For instance, the client @ 10.0.0.6 must execute:

```
$ sudo route add -net 225.0.0.0 netmask 255.0.0.0 eth1
$ sudo route add -net 224.0.0.0 netmask 255.0.0.0 eth1
```

The cNAP @ 10.0.0.2 must execute:

```
$ sudo route add -net 225.0.0.0 netmask 255.0.0.0 s8
$ sudo route add -net 224.0.0.0 netmask 255.0.0.0 s8
```

The sNAP @ 10.0.0.4 must execute:

```
$ sudo route add -net 225.0.0.0 netmask 255.0.0.0 s9
$ sudo route add -net 224.0.0.0 netmask 255.0.0.0 s9
```

We also need to force all IP user nodes (or user equipment, UEs) to use IGMPv2 (rather than IGMPv3), via the following:

```
$ sysctl -w net.ipv4.ip_forward=1
$ sysctl -w net.ipv4.conf.all.force_igmp_version=2
$ sysctl -w net.ipv4.conf.default.force_igmp_version=2
```

2.3 IGMP handler configuration

The configuration variables of the IGMP handler are included in the *nap.cfg* file. A sample configuration section for the IGMP specific parameters is given below, along with descriptive comments for each parameter of the IGMP handler. Note that these settings are the preconfigured ones for **cNAP** @ 10.0.0.2.

```
# if set "true" IGMP handler is deactivated
igmp_disableHandler = "false";
# the reserved IP for IGMP group generic queries
igmp_genQueryIP = "224.0.0.1";
# the reserved IP for IGMP group leave messages
igmp_groupLeaveIP = "224.0.0.2";
# the time period that cNAP creates and emits group generic
queries (ms)
igmp_genQueryTimer = "120000";
# the time period that cNAP waits for a response after emiting
a group
# specific query due to previous client leave (ms)
igmp_genLeaveTimer = "10000";
# the number of redundant IGMP messages to overcome network
loss
igmp_redundantIGMP = "2";
# the number of unanswered queries before group is considered
empty
igmp_dropAfterQueriesNum = "3";
# the IP addresses of served multicast groups
(serverNAP-specific)
igmp_sNapMCastIPs = "225.0.0.21:5004 225.0.0.22:5004
225.0.0.23:5004";
# the IP addresses that must be ignored
# place here also NAP's local IP address to prevent packet
loops
igmp_ignoreIGMPFrom = "172.16.7.1 224.0.0.251 224.0.0.252" ;
# NAP's local IP address to prevent packet loops
(cNAP-specific)
```

```
igmp_ignoreMCastDataFrom = "172.16.7.1";
# the NAP's operation mode: {clientNAP|serverNAP|<empty>}
# if <empty> both roles are enabled
igmp_napOperationMode = "clientNAP";
# When leaving a group, an IGMP message and THEN an unpublish message are sent.
# The processing overhead of the first can invert the sequence of the requests
# thus braking the comminicasion scheme.
# This parameter introduces a safety lag (in ns) between the two messages
# Do not leave it empty!
igmp_message_processing_lag = "500000";
```

In case node 10.0.0.2 is an sNAP, the following parameters must be updated:

```
# fill in the multicast addresses that NAP supports
igmp_sNapMCastIPs = "225.0.0.21:5004 225.0.0.22:5004 225.0.0.23:5004";
# fill in the operation mode of IGMP handler
igmp_napOperationMode = "serverNAP";
# leave empty filtering rules for loop-back multicast
# data packets - not necessary
igmp_ignoreMCastDataFrom = "";
```

2.4 Simple IGMP applications

The dummy applications are simple tools for testing and debugging the handler but also exhibit certain limitations, discussed below. These applications are coded in three source files, placed in the /POINT_HOME_DIR/examples/IGMP_streaming directory. The source files are:

- **server.c**: This file contains the server code. It implements a server that opens an ordinary UDP socket and sends data periodically (by default every second for 100 sec). The data is a string message.
- **client.c**: This file contains the client code. It implements a client that (i) binds to a port for receiving packets, (ii) sends a membership report via the Linux

IGMP kernel module (thereafter, host will reply to received group queries etc), (iii) prints incoming messages, and (iv) sends leave message when the process is killed.

- **multdef.h**: This file contains the required details of the service. It defines the channel IP address, the ports used in the communication and the message size.

2.5 Execution steps

We now present the sequence of execution steps needed to conduct a simple experiment using IP multicast. Note that the steps describing the generic operation of the POINT prototype, such as steps 1 and 3-5, are not detailed, hence readers that are unfamiliar with the required software are referred to the associated example documents.

1. Download the latest version from github - currently, the most updated github branch is “r310_PTL_IGMPHandler”
2. Update the IGMP handler configuration file
3. Update the NAP configuration file (see nap how-to file)
4. Make & install the POINT prototype
5. Create topology file and deploy testbed
6. It is advised to kill nap processes and run it manually in order to see logging messages. In the */NAP_HOME_DIR/* directory of c/sNAP run the command
    - $ sudo killall snapd; sudo nap
7. Configure IP applications
8. Start the server at the server node (e.g., 10.0.0.5)
    - $ /POINT_HOME_DIR/examples/IGMP_streaming/./mcast_ip_server
9. Start the client at the client node (e.g., 10.0.0.6 or 10.0.0.16)
    - $ /POINT_HOME_DIR/examples/IGMP_streaming/./mcast_ip_client

2.6 Restrictions

- Currently, the IGMP handler runs on a specific operational mode (server or client NAP). The adoption of both roles is possible, but not sufficiently tested, hence it is unlikely to be stable.
- Only IGMP v2 is supported.
- IGMP messages are not processed by the simple server application.

3. Advanced example

The purpose of this example is to document the creation of a functional IP multicast-over-ICN demo with IGMPv2 snooping, with realistic switching elements and end devices. The following setup was tested on KVM, with both a virtual client and a hardware STB:

1. One Linux VM used as the IPTV server with two NICs
2. One Linux VM used as a router with three NICs (see diagram below)
3. One Linux VM used as the client with two NICs
4. One sNAP and one cNAP representing the POINT network
5. (Optional) Two SDN switches and an ODL controller to demonstrate link failover with IP multicast-over-ICN.

Setting up the sNAP, cNAP, OVS switches and ODL can be performed by following the instructions in the previous sections and the HowTo-SDN.

3.1 Topology

Figure 2 shows the topology of the advanced test network, including the optional elements (OVS switches for testing path failover).

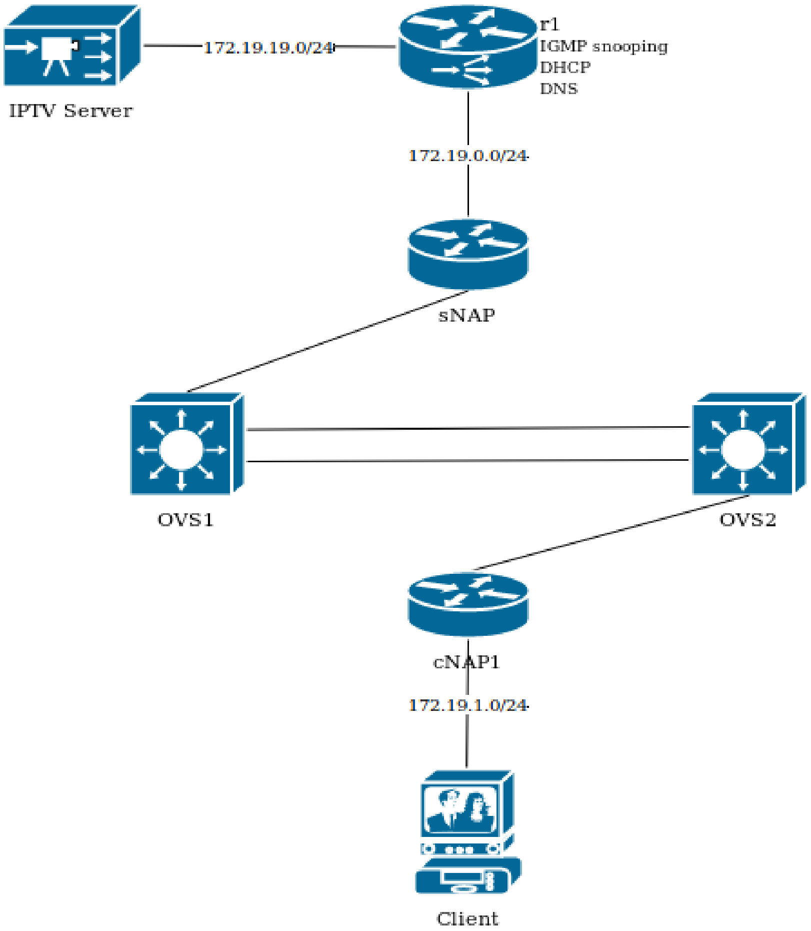

Figure 2: Advanced topology.

3.2 IPTV service router configuration

Rather than a direct connection, we will use the VyOS cloud router distribution to respresent router 1. Download the **rolling version** from https://vyos.io/ Connect to the router's console and apply the following configuration to enable IGMPv2 snooping and DHCP in `172.19.0.0/24`:

```
  conf
  set system host-name 'r1'
  set system 'disable-dhcp-nameservers'
  set system name-server '8.8.8.8'
  set system name-server '8.8.4.4'
  set system sysctl all 'net.ipv4.conf.all.force_igmp_version=2'
  set system time-zone 'Europe/Athens'
  set service ssh port '22'
  set interfaces loopback 'lo'
  set interfaces ethernet eth0 address 'dhcp'
  set interfaces ethernet eth0 description 'MGMT'
  set interfaces ethernet eth1 address '172.19.19.1/24'
  set interfaces ethernet eth1 description 'IPTV'
  set interfaces ethernet eth2 address '172.19.0.1/24'
  set interfaces ethernet eth2 description 'LAN'
  set protocols igmp-proxy interface eth0 role 'disabled'
  set protocols igmp-proxy interface eth0 threshold '1'
  set protocols igmp-proxy interface eth1 role 'upstream'
  set protocols igmp-proxy interface eth1 threshold '1'
  set protocols igmp-proxy interface eth2 role 'downstream'
  set protocols igmp-proxy interface eth2 threshold '1'
  set service dhcp-server disabled 'false'
  set service dhcp-server shared-network-name LAN authoritative 'enable'
  set service dhcp-server shared-network-name LAN subnet 172.19.0.0/24
default-router '172.19.0.1'
  set service dhcp-server shared-network-name LAN subnet 172.19.0.0/24 dns-server
'172.19.0.1'
  set service dhcp-server shared-network-name LAN subnet 172.19.0.0/24
domain-search 'local'
  set service dhcp-server shared-network-name LAN subnet 172.19.0.0/24 lease '10'
  set service dhcp-server shared-network-name LAN subnet 172.19.0.0/24 start
172.19.0.100 stop '172.19.0.199'
  commit
  save
  exit
```

Static routes are also needed for the example:

```
set protocols static route 172.19.1.0/24 next-hop '172.19.0.2' distance 1
```

3.3 IPTV content server configuration

We will use `ffmpeg` to stream content. Connect the IPTV server machine to the IPTV network interface of router 1, eth1:

```
auto ens4
```

```
iface ens4 inet static
    address 172.19.19.20
    netmask 255.255.255.0
    broadcast 172.19.19.255
    gateway 172.19.19.1
```

Download some demo content (e.g. http://support.aminocom.com/ics/support/DLList.asp?folderID=9244) and use `ffmpeg` to transcode the file to the proper MPEG-TS format compatible with multicast streaming over UDP:

```
ffmpeg -i big_buck_bunny_pal.ts -vcodec mpeg2video -s 1280x720 -r 25 -flags
cgop+ilme -sc_threshold 1000000000 -b:v 3M -minrate:v 3M -maxrate:v 3M -bufsize:v
1.8M -acodec mp2 -ac 2 -b:a 128k -f mpegts bbb_720p3M.ts
```

Use `ffmpeg` to stream the file to a multicast group over UDP:

```
ffmpeg -re -i bbb_720p3M.ts -codec:v copy -codec:a copy -f mpegts
udp://239.192.0.13:1234?pkt_size=1316?fifo_size=10000
```

3.4 Client configuration

Connect a client VM to the LAN of `r1` or the cNAP and set either up as the client's default gateway. Issue:

```
vlc udp://@239.192.0.13:1234?ttl=2
```

or

```
ffplay udp://239.192.0.13:1234?ttl=2
```

Monitor the traffic on router 1:

```
monitor interfaces ethernet
```

Expected results: `eth1` on router 1 should be ingesting traffic at around 3 Mbps. Traffic on `eth2` should be near zero, unless the client requested to play the multicast stream. Using `ffmpeg` with the parameters above, the video should be playing smoothly on the client.