Test to Python implementation of IGMPv2, PIM-DM, and HPIM-DM

Pedro Francisco Carmelo de Oliveira
pedro.francisco.oliveira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

March 24, 2019

In order to check if the implementations were according to the specifications, some tests were done. All tests were performed using Netkit-NG [1], allowing to emulate a network environment. Additionally, we have also used GNS3 [2] to verify the interoperability between our implementations executing in Linux devices and the implementations of Cisco routers.

A test is defined by a certain topology in order to verify if a certain module of the protocol is working as expected. All packets exchanged between routers were captured using *tcpdump* [3] and further analyzed with *Wireshark* [4].

All tests were performed by knowing a priori the state in which some routers should be at. A centralized node knows this information and all routers send their state transitions to it. A test is considered to be successful when all predicted state transitions were made. This was accomplished by sending logs of our implementation to a server node, according to state transitions, exchange of control messages and expiration of timers. The remote logging was accomplished by the Python's *logging* module, that has an handler that can transmit all recorded logs to a given node. In order to do this, all routers would execute the Test command that was referred in the Python implementation [5]. Regarding the exchanged control messages, these were manually verified with *Wireshark* [4] after executing the tests.

We will describe all performed tests and corresponding results to our IGMPv2, PIM-DM and HPIM-DM implementations.

- **Section 1** describes all tests to our IGMPv2 implementation;
- **Section 2** describes all tests to our PIM-DM implementation;
- **Section 3** describes all tests to our HPIM-DM implementation.

1 IGMP Implementation Test Results

In order to test IGMP, we only need one subnet with one or more routers and one or more hosts connected to it. The used topology is shown in Figure 1. Since routers did not require to communicate

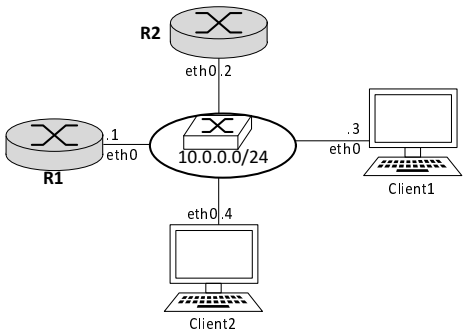

Figure 1: Topology used to test IGMP, PIM-DM and HPIM-DM implementations.

beyond this local subnetwork, it was not configured any unicast routing protocol on routers.

Hosts manifested their interest, reporting and leaving groups, by opening and closing sockets. The host part of IGMPv2 was totally controlled by the Linux implementation of the network devices.

One Linux machine had the job of a switch, in order to connect more than one router to a single subnet. This was possible by executing special commands, such as "brctl" on that Linux machine. This command simply allows to aggregate multiple physical interfaces into one logical interface, belonging all to the same subnetwork.

The server process that received all logs was running on the switch, which printed to the console all received logs regarding IGMPv2 (of routers) and stored them in a file.

The switch also captured all exchanged packets, using *tcpdump*, by executing the command "tcpdump -i br0 -Q in -w Testx.pcap". This command recorded all packets that entered interface br0, which was the logical bridge interface that aggregated all physical interfaces of the switch, and wrote those packets to a capture file named Testx.pcap, being x the number of the test.

The implementation of the IGMPv2 protocol was integrated in both PIM-DM and HPIM-DM implementations. Since the IGMPv2 implementation was not intended to be used alone, all tests were performed in HPIM-DM implementation. Also, all implemented timers were set to their default values, according to the IGMPv2 specification [6].

We have subdivided the tests of IGMP in two parts. The first part tested if all routers elected the Querier correctly and if that router transmitted the expected control messages. The second part tested if all routers were determining correctly the interest of hosts in a particular group.

The Netkit network configurations, tests and corresponding results are stored in a branch named "Test_IGMP" at our GitHub repository [7].

1.1 Querier election and maintenance

First we will describe the test results regarding the Querier state machine:

- **Test 1 -** Election of the Querier, after two routers start the IGMP process

  In this test we started the IGMP protocol first on router R1 and then on router R2. We have verified that both routers initially transmitted IGMP General Membership Queries, destined to the All-Systems multicast IP address (224.0.0.1). Both routers started initially in Querier state and since router R1 transmitted its control message when R2 had not started its protocol, this meant that R2 did not interpret that message. For this reason, both routers remained in the Querier state, until R1 retransmitted an IGMP General Membership Query.

  When R1 retransmitted an IGMP General Membership Query, controlled by the "general query timer", R2 by hearing this message, transitioned to Non-Querier state. From this point on, R2 did not transmit anymore IGMP General Membership Query messages.

  It was verified in Wireshark that all messages were correctly built, by being properly dissected. Also, it was possible to check the periodicity of IGMP Membership Query message transmissions, which was around 125 seconds. This periodicity was according to the specification.

  With this test, it was possible to verify that the Querier state machine made the correct transitions and correctly elected the Querier.

- **Test 2 -** Reelection of the Querier after the previous one fails

  We have repeated Test 1, but now we have started first the IGMP process in R2 and only then in R1. This allowed a quicker transition to the correct state (Non-Querier) at R2.

  We have waited for the first IGMP General Membership Query retransmission of R1 and then stopped the protocol process at R1. This message was transmitted at time 131 seconds, according to Wireshark.

  After a given amount of time, in the logs it was possible to verify that the "other querier present timer" of R2 had expired, causing a transition to the Querier state in this same router. Then an IGMP General Membership Query was transmitted by R2 at time 386 seconds. The delta between the last transmission of R1 and the first transmission of R2 was around 255 seconds, which was according to the specification ((the Robustness Variable) x (the Query Interval) + (one half of one Query Response Interval) = 2 x 125 + 0,5 x 10 = 255).

With this test it was possible to verify that the implementation of IGMP correctly reacted to failures of the Querier, due to the absence of its IGMP Membership Query messages. Also, the timer used to control this state was set correctly.

- **Test 3 -** Failure of a Non-Querier must not cause a reelection

  We have started the protocol first in R2 and then in R1, like described in Test 2. Also, we have waited for the first retransmission of an IGMP Membership Query by R1 and then stopped the protocol at R2.

  In this test it was possible to verify that R1 remained in the Querier state, not being affected by the failure of R2. This test and Test 1 are indistinguishable in terms of router R1, since it remained in the Querier state.

  The periodicity of exchanged control messages was according to the previous tests and to the specification.

1.2 Membership maintenance

Now, we will describe the test results regarding membership detection by routers:

- **Test 4 -** Detection of membership when hosts are not interested in receiving multicast traffic (all groups remain in "No Members Present" state)

  This is similar to all tests that were performed until now (from Test 1 to 3), since on those tests hosts did not manifest their interest in receiving data packets.

  On those tests we did not observe any IGMP Report or IGMP Leave message. Also, by manually verifying the list of groups that the protocol process was monitoring, we found that list to be empty. This was expected, since the process only starts monitoring a given group when it receives an IGMP message regarding it. By having the list empty, there are no groups being monitored, which means that all groups were considered to be in a "No Members Present" state.

- **Test 5 -** Detection of membership when a previously not interested group starts having interested hosts (hosts were not interested in receiving multicast traffic but one of them declares interest)

  Two hosts were not interested in any group but one of them declares interest in group "224.12.12.12". It is expected that all groups remain in "No Members Present" state, except for the reported group that will transition to "Members Present", in both routers.

  In this test the two routers, R1 and R2, were running the protocol process and Client1 declared interest in group 224.12.12.12, by transmitting an IGMP Membership Report message.

  We have observed an unexpected behavior regarding the transmission of IGMP Report messages from the host. We have observed three consecutive transmissions of this type of message, when the specification only specifies two. The additional transmission may be for reliability purposes. Since we are only interested in testing the IGMP implementation of routers, we were not concerned with this unexpected behavior, since it does not cause any malfunction to the protocol.

  By hearing the first IGMP Membership Report, both routers transitioned group "224.12.12.12" to "Members Present" state. From time to time, the Querier transmitted IGMP Membership Query messages, which were answered with an IGMP Membership Report regarding group "224.12.12.12" by the same host.

  When the Querier transmits an IGMP Membership Query message, it specifies a given amount of time that it will wait for the reception of IGMP Report messages. This amount of time is called Max Response Time and is included in a field of the transmitted IGMP Query message. The default amount of time, according to the specification, is 10 seconds. It was observed that the host responded within this time interval. The first Report message was transmitted 2 seconds after the Query message has been received, while the second Report was transmitted 4 seconds after the second Query message was received and a third Report was transmitted 8 seconds after the third Query message has been received. The amount of time a host waits before it transmits a Report is chosen randomly, in order to support the suppression mechanism, i.e. a host does not require to transmit a Report if other host has already reported the same group.

Since both routers remained in "Members Present" state in group "224.12.12.12", we can conclude that the reporting of membership was correctly detected. Also, the field Max Response Time and the corresponding timer were set correctly.

- **Test 6 -** Detection of membership when a previously interested group starts having interest by another host (an host manifested interest regarding a group that already had interested hosts)

  Two hosts were not interested in any group but one of them declares interest in group "224.12.12.12". Then the second host reports interest in the same group. It is expected that all groups remain in "No Members Present" state, except for that reported group, which will transition and remain in "Members Present" state, after the first host reports it.

  In this test the two routers were running the protocol process and both hosts reported interest in group "224.12.12.12".

  Like Test 5, we have observed that both hosts transmitted initially three consecutive IGMP Report messages and then only retransmitted this message after hearing an IGMP Membership Query message.

  By looking to the packet capture, it looked unexpected at first sight because both hosts were "Reporting" the same group after hearing IGMP Membership Query messages, suggesting that the suppression mechanism was not working. This was due to a technology built-in in some switches called IGMP Snooping. Basically this allows the switch to interpret IGMP messages and define which interfaces of the switch have interest in receiving multicast traffic. This allows multicast traffic to not behave like broadcast traffic in a local network. For this reason, IGMP Membership Report messages transmitted from a switch port were not received by a host connected to other switch port. This requires to specify which interfaces of a switch have multicast routers connected to it, in order to receive all multicast traffic and all IGMP message types. Since we were only interested in testing the router's IGMP state machines, this suppression mechanism is out of scope because it is only used at the host part.

  The difference between this test and Test 5 is that two hosts Report group "224.12.12.12" instead of just one. Like the other test, all hosts reported within the Max Response Time, which allowed all routers to remain in the correct state regarding this group ("Members Present").

- **Test 7 -** Detection of membership when an host leaves a group in which there is still interest by other hosts

  Two hosts declared interest in group "224.12.12.12" and one of them leaves it, by transmitting an IGMPv2 Leave message. Since the other host remains interested in the same group, it is expected that this group remains in "Members Present". Since there was a Leave message involved, first all routers will transition to "Checking Membership" state and then transition again to "Members Present".

  In this test the two routers, R1 and R2, were running the protocol process and Client1 and Client2 "Reported" initially interest in the same group "224.12.12.12". After a given amount of time, Client2 left group "224.12.12.12", by transmitting an IGMP Leave message, regarding it.

  In the logs we can observe that both routers received this message. Router R2 did not react to this message, because it was in Non-Querier state. Router R1 by receiving the same message, transitioned to state "Checking Membership" state and transmitted an IGMP Group-Specific Query. Only the reception of this last message by R2 caused this router to transition to "Checking Membership" state.

  After a given amount of time, the host still interested in group "224.12.12.12", responded to the IGMP Group Specific Query, by transmitting an IGMP Membership Report message. In the logs it was possible to verify that the reception of this message caused both routers to transition group "224.12.12.12" back to "Members Present" state.

  The transmitted IGMP Group Specific Query had Max Response Time of 1 second, like specified in the specification (Last Members Query Interval = 1 second). The response from the interested host was within this time interval, being observed a difference of 0.2 seconds between the reception and the transmission of those messages.

All state transitions were expected and the routers did not transition to "No Members Present" during this process. The implementation reacted accordingly to the received messages and the timers were set with the expected values.

- **Test 8 -** Detection of membership when an host leaves a group and there are no other hosts interested in this same group

  Initially only one host is interested in group "224.12.12.12", then stops being interested. Since there are no more hosts interested on this group, it is expected a transition to "No Members Present" state by both routers.

  We have started by recreating Test 7, i.e. two hosts initially interested and then one host leaves the group. Then we have observed what happened when the last interested host leaves that same group.

  When the last host leaves group "224.12.12.12", by transmitting an IGMP Leave message, we have observed the same initial transition to "Checking Membership" state and the transmission of an IGMP Group Specific Query.

  Since there was no response from the hosts to the IGMP Group Specific Query, within the Max Response Time of 1 second, the Querier retransmitted this message (due to the expiration of the retransmit timer). By not hearing any response from the hosts to the second IGMP Group Specific Query, both routers transitioned group "224.12.12.12" to "No Members Present" state. This transition, in both routers, was caused by the expiration of the timer, which was possible to verify in the logs.

  As it was expected, both routers correctly determined when a given group had no members interested in receiving data packets.

- **Test 9 -** Detection of membership when an host that manifested interest fails, without leaving the group through an IGMP Leave message

  One host declares interest in group "224.12.12.12" and then fails, without transmitting an IGMPv2 Leave message. Since the "Members Present" state is only maintained for a given amount of time, it is expected that eventually all routers transition to the correct state ("No Members Present"), after the timer associated with it expires.

  In this test, only one host reported interest in group "224.12.12.12", which was Client1. We started by initializing the protocol in both routers, R1 and R2, and waited for them to correctly transition to their Querier/Non-Querier states. Then Client1 reported interest in group "224.12.12.12", causing a transition of this group to "Members Present", in both routers.

  Then, after three successive IGMP Membership Queries and corresponding IGMP Membership Reports, we simulated the failure of Client1 by shutting down its interface that connected this host to the local network.

  It was possible to verify in the logs that after a given amount of time, both routers transitioned correctly to the "No Members Present", due to the expiration of the timer regarding group "224.12.12.12". This expiration happened around 4 and a half minutes after the last IGMP Membership Report was received, which is consistent to the expected time. This time is defined in the specification by Group Membership Interval = (Robustness Variable) x (Query Interval) + (Query Response Interval) = 2 x 125 + 10 = 260 seconds = 4.333 minutes.

  With this test, it was possible to verify that the absence of IGMP Report messages was correctly detected. This caused both routers to no longer consider the group to have members. Also, the timer used to control this behavior was set according to the specification.

2 PIM-DM Implementation Test Results

Since PIM-DM requires information from the unicast routing table, in order to form and shape multicast distribution trees, routers need to run a given unicast routing protocol or configure statically all subnetworks. It was opted to use OSPF [8] as the underlying unicast routing protocol. For this reason all Linux routers executed *Quagga* [9], which offered an implementation of this protocol.

2.1 Neighborhood maintenance

First we will describe the test results regarding the establishment and maintenance of neighborhood relationships. These were performed using the topology of Figure 1.

The corresponding Netkit network configurations, tests and results are stored in a branch named "Test_PIM_Hello" at our GitHub repository [10].

The timer associated with the transmission of Hello messages was set to its default value, i.e. transmission periodicity of 30 seconds. In case a new or rebooting neighbor is detected, a new Hello message is transmitted within 5 seconds (randomly). The packet captures must confirm these values.

Also, State Refresh was not enabled on these tests, since we were only testing if two routers correctly established a neighborhood relationship. By having this feature disabled, the State Refresh option was not present in the transmitted Hello messages.

- **Test 1 -** Establishment of neighborhood relationship between two unknown neighbors

  In this test we have two routers, R1 and R2, directly connected to each other. We have first started the protocol process in R1 and then on R2.

  In the logs, we can confirm that both routers established a neighborhood relationship due to the reception of an Hello message from the corresponding neighbor.

  In the packet capture, we can verify that R1 transmitted Hello messages having 105 as the Hello Hold Time and Generation ID set to 1217533332. R2 also transmitted Hello messages with the same Hello Hold Time and Generation ID set to 3312157476. Following messages by both routers transmitted the same options and values, as it was supposed. In the logs, we can verify that both routers correctly interpreted these values.

  The transmitted Hello Hold Time was correct, in both routers, since this value is set to 3.5 times the periodicity of Hello messages (default of 30), being 3.5*30=105 seconds.

  Regarding the message periodicity, according to Wireshark, when both routers started detecting each other, they transmitted Hello messages within the expected range of 5 seconds (R2 transmitted at 15.35 seconds due to the boot up; R1 transmitted at 19.89 seconds due to the discovery of R2; R2 transmitted at 20.88 seconds due to the discovery of R1). Then after discovering each other, they transmitted Hello messages every 30 seconds. R1 transmitted at 19.89 seconds, 49.93 seconds, 79.99 seconds, ..., while R2 transmitted at 20.88 seconds, 50.92 seconds, 80.98 seconds, ...

  According to this test, both routers correctly interpreted the exchanged Hello messages and established a neighborhood relationship. Also, message transmissions were within the expected time.

- **Test 2 -** Re-establishment of a neighborhood relationship after a known neighbor reboots

  We started by recreating Test 1, i.e. start the protocol process in both routers, in order to establish a neighborhood relationship. Note that since we have restarted the protocol process in both routers, they transmitted Hello messages with different Generation IDs (R1 with Generation ID set to 3164982407 and R2 set to 3806743712) compared to the previous test. This was normal since these values change every time the protocol starts/restarts in a given interface.

  After R1 and R2 established a correct neighborhood relationship, we rebooted the protocol process in R2. This caused R2 to transmit Hello messages with a newer Generation ID, set to 1471879794.

  In the logs we can observe that the reception of an Hello message with a different Generation ID, caused R1 to correctly detect the reset of R2.

  The periodicity of Hello messages was within the expected values, like it was observed in the previous test. Also, it was possible to determine that a router correctly interpreted a neighbor reset.

- **Test 3 -** Neighborhood relationship break after known neighbor fails

  In this test, we started by recreating Test 1 in order to have both routers, R1 and R2, with a established neighborhood relationship. Like Test 2, since we have restarted the protocol process, both routers transmitted Hello messages with different Generation IDs, compared to the previous tests.

  After both routers established a neighborhood relationship, we waited around 60 seconds and stopped the protocol process in R2. This caused R2 to no longer transmit Hello messages.

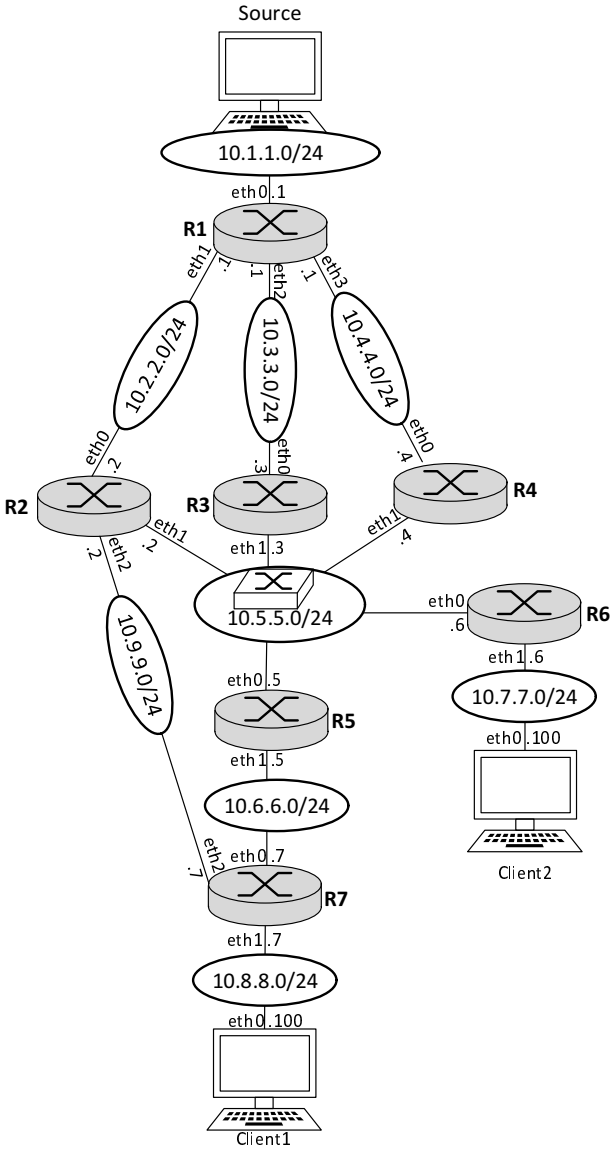

Figure 2: Topology used to test the creation and maintenance of the trees in both PIM-DM and HPIM-DM implementations.

According to the logs, R1 received the last Hello message from R2 at time 18:02:21. At time 18:04:06, R1 considered R2 to have failed due to the absence of received Hello messages. This time interval is of about 1 minute and 45 seconds, which represents 105 seconds. It was possible to verify that the detection of a neighbor fail was detected correctly, since the last transmitted Hello message from R2 had the Hello Hold Time option set to this precise time.

In this test we could verify that a neighbor failure was detected correctly. Also the timer used to control the liveness of a known neighbor was set correctly.

2.2 Formation and maintenance of the broadcast tree

Now we will describe the test results regarding the formation of the broadcast tree in order to verify if it was being constructed correctly, according to the unicast routing table. These were performed in the topology represented by Figure 2. In order to accomplish these tests, we first needed to verify if the unicast routing table had already stabilized.

Regarding the unicast routing protocol, each interface was set with a default cost of 10, meaning that the root and non-root interfaces were selected according to the "number of hops", in the sense that paths

with fewer hops were preferred due to having a lower Root Path Cost. For this reason, the root interface, according to the subnet of Source, must be interface eth0 in all routers except for Router7 that must be eth2.

The corresponding Netkit network configurations, tests and results regarding the formation of the broadcast tree are stored in a branch named "Test_PIM_BroadcastTree" at our GitHub repository [10].

- **Test 4 -** Formation of the broadcast tree (flood of data packets and selection of root interface)

  In order to test this, the two receivers manifested interest regarding group "224.12.12.12". Host represented by Source transmitted multicast data packets destined to this group. In order verify if the broadcast tree was correctly formed, the two hosts must receive those multicast data packets and the multicast routing table, in all routers, must be correctly set (i.e. their root and non-root interfaces).

  When the unicast routing table at all routers had information regarding all subnets, Source started transmitting data packets to group "224.12.12.12".

  By looking to the logs, we have verified that all types of interfaces were chosen correctly (root and non-root). Also, multicast data packets were received in both clients. For this reason, it was possible to verify that the broadcast tree regarding (10.0.0.1, 224.12.12.12) was correctly built.

  Regarding the packet captures, we ran *tcpdump* in all links. It was possible to verify that UDP data packets originated from Source and destined to group 224.12.12.12 were initially present in all captures, confirming that the broadcast tree was formed due to the flood of multicast data packets.

  Control messages exchanged by routers were not verified in this test, since those will be further analyzed on following tests and because the creation of the broadcast tree is only triggered by the reception of data packets.

- **Test 5 -** Broadcast tree reconfiguration in reaction to RPC changes detected by the unicast routing protocol.

  To perform this test, we selected interface eth2 of Router7 to suffer an RPC change. This must cause a change in the interfaces' roles of this router (root <-> non-root interface). Since all interfaces have a default cost of 10, we have changed the cost of interface eth2 of Router7 to 100, in order for eth0 to be selected as the new root interface.

  By analyzing the logs, we have verified that interface eth2 of R7 transitioned to non-root and interface eth0 transitioned to root, as expected.

  With this test, we have verified that the protocol correctly reacted to changes at the unicast routing table, reshaping the already formed (10.0.0.1,224.12.12.12) tree.

- **Test 6 -** Broadcast tree reconfiguration in reaction to router failures, detected by both the unicast routing protocol and the multicast routing protocol.

  After performing Test 5, without restarting the protocol in any router, the broadcast tree was set in all routers, with eth0 being set as the root interface on all routers. In this test, we have opted to fail router R5 in order to verify if the root interface of router R7 changes again to eth2.

  In order to "simulate" the failure of a network device, we have disabled (shutdown) interface eth0 of R5. Eventually, the unicast routing protocol reacted to this change, modifying the unicast routing table of some routers. This modification was visible in the logs, with the change of the root interface at R5 and R7.

  With this test, we have verified that all interfaces were set correctly, even when the unicast routing protocol suffers changes due to router failures.

With all performed tests regarding the creation of the broadcast tree, it was possible to verify that the selection of root and non-root interfaces was performed correctly. Also, by analyzing the packet captures, it was possible to verify that multicast data packets were initially exchanged in all links, like it was supposed to, due to the flooding behavior of this protocol.

2.3 Assert mechanism

Now we will describe the test results regarding the Assert mechanism. These tests were performed on the same topology of the previous tests, represented by Figure 2. In theses tests, we only focused on what happened on the subnet that connected routers R2, R3, R4, R5 and R6. The corresponding Netkit network configurations, tests and results are stored in a branch named "Test_PIM_Assert" at our GitHub repository [10].

- **Test 7 -** Election of a single Assert Winner

  The flood of multicast data packets from host Source, must trigger the Assert mechanism, causing the exchange of Assert message with RPC information. Since all routers have their interfaces set with default costs, routers R2, R3 and R4 have the same RPC to Source, causing the Assert mechanism to use the IP address to perform the election. It was expected that router R4 would won the election, since its non-root interface had the greatest IP address.

  In the packet capture, we have verified that routers started exchanging Assert messages after they forward the first multicast data packets. We have observed a triplication of the same initial data packet, due to this being transmitted by R2, R3 and R4. The Assert messages, transmitted after the first data packet, were well dissected and had the same RPC (20), as expected.

  It was also possible to verify that in reaction to the Assert messages, downstream routers transmitted Graft messages whenever a new Assert Winner was elected. We have observed that routers R5 and R6 transmitted a Graft message destined to R2 because this router was the first one to transmit an Assert message, which was initially elected as the Assert Winner by these two routers. Then R4 transmitted an Assert message and by having a greater IP address than R2, both downstream routers considered a change of the Assert Winner, causing the transmission of a new Graft message but now destined to R4. Then R3 transmitted an Assert message but since this message lost the Assert compared to the already elected Assert Winner (R4), downstream routers did not transmit new Graft messages. These Graft messages were acknowledged back with Graft-Ack messages.

  Also, whenever an upstream router lost the Assert mechanism, it transmitted a Prune due to the reception of an Assert with the same RPC but a greater IP address, which was observed in the packet capture. Routers R2 and R3 transmitted Prune messages due to the reception of the Assert from R4.

  In the log, we have observed that all routers connected to the switch, transitioned from NoInfo to the Loser state, except for router R4 that transitioned to the Winner state.

  With this test we have verified that the implemented Assert mechanism elected correctly an Assert Winner. Also, routers reacted correctly to this election, by transmitting Prune messages whenever a non-root interface lost the Assert and by transmitting Graft messages whenever a root interface detected a change of the Assert Winner.

- **Test 8 -** Reaction to the failure of the current Assert Winner (without having data packets being forwarded)

  After Test 7, we have stopped the protocol process only in interface eth1 of the Assert Winner (R4). This caused the interface to transmit an Hello message with 0 as the Hello Hold Time, causing all routers to stop considering R4 as being "alive". Since this router was considered to be the Assert Winner, all routers reacted to its failure by transitioning to the NoInfo state. In the logs, we have observed this expected behavior.

  It was not transmitted any data packet during this test, which caused all routers to remain in NoInfo state. This proved that the implementation reacted correctly to the removal/failure of the Assert Winner.

- **Test 9 -** Reelection of the Assert Winner, after the previous one fails, with data packets being forwarded

  After Test 8, the transmission of data packets should reelect a new Assert Winner (R3).

  The host Source by transmitting data packets, caused those data packets to be transmitted by routers R2 and R3. Since data packets were received in non-root interfaces, this triggered the transmission of new Assert messages by both R2 and R3.

The Assert messages had the same RPC to the Source subnet, like it was supposed to. This caused the election of the Assert Winner to be accomplished by the comparison of the IP address. Since router R3 had a greater IP address, it was elected as the Assert Winner, causing a transition to the Winner state. All the other routers transitioned to the Loser state, in reaction to the exchanged messages. This behavior was observed in the logs and in the packet capture.

After the exchange of data packets and Assert messages, the router that lost the Assert (R2) transmitted a Prune message, like it was supposed to. Also, in reaction to the new Assert Winner (R3), downstream routers sent Graft messages destined to it. These messages were acknowledged back with a Graft-Ack message.

This test proved that after a failure/removal of the Assert Winner, all non-root interfaces by transmitting data packets, triggered a new Assert Winner election with the exchange of new Assert control messages.

- **Test 10 -** Reelection of the Assert Winner in case the previous one changes its interface role (becomes root type)

  After Test 9, R3 was elected as the Assert Winner of the link. We have changed the cost of interface eth0 of this router to 100, in order for interface eth1, of this router, to become root type.

  The change of the interface cost and corresponding change of its role caused the transmission of an Assert Cancel message via interface eth1 of R3, as it can be observed in the packet capture. This caused all routers to transition to the NoInfo state, as it was observed in the logs. This Assert Cancel message represents an Assert message with RPC of infinite.

  Following data packets were forwarded by router R2 due to its transition to the NoInfo state.

  With this test, we have verified that the implementation reacted correctly to the change of the interface role at the Assert Winner, by having all routers transitioning to the NoInfo state with the reception of an Assert Cancel message from the current Assert Winner.

With these tests, we have verified that the Assert Winner was elected correctly. This election was performed correctly even when the current winner fails and whenever its interface becomes root type.

2.4 Pruning, Joining and Grafting

Now we will describe the test results regarding the Pruning, Joining and Grafting of the tree and if the Prune Override mechanism is not pruning branches of the tree that have routers still interested in receiving data packets. To test this, we used the same topology of the previous tests, represented by Figure 2. In this topology, we have explicitly set interface eth2 of Router7 to have a cost of 100. This way, interface eth0 was the root interface of Router R7, allowing to test if the Prune Override mechanism, at the switch, was working correctly, being controlled by the membership of both Client1 and Client2. In order to perform these tests, we have captured all packets exchanged at all links of the network topology, in order to verify if the correct control and data packets were exchanged. These tests will focus on the Upstream Interface and Downstream Interface state machines.

The corresponding Netkit network configurations, tests and results are stored in a branch named "Test_PIM_JOIN_PRUNE_GRAFT" at our GitHub repository [10].

To perform these tests, all timers were set to their default values, meaning that a non-root interface after hearing a Prune message will remain in PrunePending state for 3 seconds (controlled by the PrunePending timer), in order to give time for downstream routers to override the Prune. Downstream routers that want to override the Prune with a Join message, must perform this within 2.5 seconds (Override timer). The time set for the Override timer is randomly chosen in order to support the suppression mechanism (a router does not need to send a Join if other downstream router has already transmitted that same message).

- **Test 11 -** Pruning of redundant paths from the broadcast tree

  In this test, we started by having Client1 and Client2 interested in receiving multicast data packets destined to group 224.12.12.12. This way, we could test if redundant paths at the broadcast tree were pruned from it.

  Host Source by transmitting its data packets, these got flooded on the network, which was possible to verify in all packet captures.

We have observed the following state transitions at all root and non-root interfaces, and we verified that these were correct:

- R1: eth0 set to Forwarding state in Upstream Interface state machine; eth1 set to Pruned state in Downstream Interface state machine; eth2 set to Pruned in Downstream Interface state machine and eth3 set to NoInfo in Downstream Interface state machine.

  The Pruned interfaces were due to the routers that were directly connected to those interfaces, R2 and R3, have lost the Assert mechanism at the switch.

  Interface eth3 was in NoInfo state because it was connected to router R4 that won the Assert and had downstream routers interested in receiving those data packets.

  Interface eth0 was in Forwarding state because one non-root interface (eth3) must forward data packets received by the root interface.

- R2 and R3: eth0 set to Pruned state in Upstream Interface state machine.

  The state of non-root interfaces at these routers was not relevant, since what defined if they must forward multicast data traffic was the Assert state machine. These interfaces could be placed in Pruned/NoInfo state depending on the last received control message.

- R4: eth0 set to Forwarding state in Upstream Interface state machine and eth1 set to NoInfo in Downstream Interface state machine.

  Interface eth1 was placed in NoInfo state since it connected to routers that reached interested receivers. For this reason, it was placed in this state.

  Interface eth0 was placed in Forwarding state because one non-root interface must forward data packets (eth1).

- R5: eth0 set to Forwarding in Upstream Interface state machine and eth1 set to NoInfo state in Downstream Interface state machine.

  Interface eth1 was set to NoInfo state because it connected to one interested router (R7). For this reason, interface eth0 was set to Forwarding state.

- R6 and R7: eth0 set to Forwarding in Upstream Interface state machine.

  Interface eth0 was set to Forwarding state in these routers because they must forward data packets, due to one of their non-root interfaces being directly connected to interested receivers.

  The state of interface eth1, on these routers, was not relevant for this test, because they were not connected to neighbor routers, so they remained in NoInfo state. Regarding interface of eth2 of R7, the forwarding of data traffic was controlled by the Assert state machine, for this reason its state was not relevant for this test.

Regarding exchanged control messages, we have verified that the link that connects R2 to R7 has exchanged a Prune message, due to the Assert mechanism (R7 lost the Assert).

In the switched network, downstream routers transmitted Graft messages whenever a new Assert Winner was elected, like referred in the Assert tests. These messages were acknowledged back with a Graft-Ack message. This was supposed to happen since a new Assert has been elected and downstream routers were interested in receiving data packets (due to the interest of Client1 and Client2).

Also in the switched network, we have observed that Assert Losers transmitted Prune messages. This caused the Prune Override mechanism to be triggered, since downstream routers were still interested in receiving data packets. We have observed that the last transmitted Prune message had a response from R5 with a Join message. This time difference between the transmission of the Prune and the Join was of 1 second, which was within the time range of the default time (2.5 seconds).

The two clients have received all transmitted data packets from Source. The initial transmitted data packets were triplicated, due to the three upstream routers connected to the switched network, that flooded the first data packet. Then, after the Assert, transmitted data packets were received correctly without duplications.

With this test, we have verified that all routers transitioned to the correct state regarding their Upstream and Downstream Interface state machines, when the tree flooded for the first time.

- **Test 12 -** Correct multicast forwarding decision in case Client2 leaves the group

  Client2 by leaving the group, this should cause the pruning of the multicast distribution tree. The Prune Override mechanism should not prune the switched network due to the interest of Client1.

  We have started by closing the socket of Client2, in order for it to send an IGMP Leave message.

  Regarding the state transitions, we have observed that interface eth0 of router R6 was placed in Pruned state, in reaction to the IGMP Leave of Client2. This caused R6 to transmit a Prune message.

  The Prune Override mechanism was triggered, causing router R4 to transition to PrunePending state. We have observed that router R5 transmitted a Join message in reaction to the expiration of its Override timer. This expiration happened, according to the logs, 2 seconds after the reception of the Prune message, which was within the range of 2.5 seconds specified by the protocol.

  The transmission of the Join, by R5, caused the interface of the Assert Winner (R4) to transition back to the NoInfo state.

  With this test, we have observed that the Prune Override mechanism worked correctly. Also, the timers used to control this mechanism were set accordingly.

- **Test 13 -** Correct multicast forwarding decision in case both hosts leave the group

  Client1 by leaving the group must cause the pruning of the tree since there is no host interested in receiving data packets. After Test12, Client1 left group 224.12.12.12 in order to not have any member interested in this traffic.

  We have observed, according to the logs, that previously non-Pruned branches transitioned to a Pruned state (interfaces that connect R7 to R5, R5 to R4 and R4 to R1).

  According to the *Wireshark* capture, in the switched network, the reception of the Prune, from router R5, caused R4 to transition to a PrunePending state. Since no router was interested in receiving multicast data packets, no one did override the Prune with a Join message. Eventually, the PrunePending timer of R4 expired, causing this router to transition to the Pruned state. According to the logs, the expiration of this timer happened 3 seconds after the reception of the Prune message, like specified.

  After the expiration of the PrunePending timer at R4, which caused interface eth1 of this router to transition to the Pruned state, we have observed in the packet capture, that this router, Assert Winner of the switched network, transmitted a Prune message. This message is known as Prune Echo message and has the goal of adding extra reliability to the Prune Override mechanism. Since no one has transmitted a Join after this message, the router remained in the Pruned state.

  With this test, we have confirmed that the protocol reacted accordingly to the removal of interested members, causing the pruning of the whole tree. Also, the Prune Override mechanism pruned the tree due to the absence of a Join message. We have also observed the transmission of a Prune Echo message from the current Assert Winner, adding extra reliability guarantees.

- **Test 14 -** Correct multicast forwarding decision by having the tree "grafted" (Client2 becomes again interested in receiving data packets)

  Client2 by being again interested in the group must cause the "grafting" of the tree. After Test 13, Client2 became interested again in receiving data packets of group 224.12.12.12. According to the logs, this caused router R6 to transition to AckPending state. In the *Wireshark* capture we can verify that a Graft message destined to R4 was transmitted by R6.

  Also in the logs, the reception of the Graft message, by R4, caused this router to transition its non-root interface to NoInfo state. In the *Wireshark* capture, we can observe that R4 acknowledged back with a Graft-Ack message, destined to router R6. In the logs, we can observe that the reception of the Graft-Ack message, by R6, caused this router to transition from an AckPending state to a Forwarding state, as expected.

  Router R4, by being connected to downstream routers, caused a graft of the tree via its root interface (eth0), in order for traffic originated from Source to be forwarded from router R1 to R4 again. The same Graft and Graft-Ack exchange happened at that link.

  Then Source transmitted data packets and we have correctly verified that those packets reached the new interested receiver.

With this test, we have verified that the tree was correctly grafted, in order to forward data packets through previously pruned links.

- **Test 15 -** Correct multicast forwarding decision in reaction to the timeout of pruned branches (re-flood of multicast data traffic after this timeout)

  After Test 14, we have waited a given amount of time. We have verified in the logs that approximately at time 16:05:00 all routers transitioned to state NoInfo, in reaction to expiration of timers and Graft messages being exchanged. The first test was started at time 16:02:00, which confirmed that the pruned tree only lasted three minutes, like defined in the specification. In the logs, we can verify that this happened due to the expiration of the timers that control the Assert and Downstream Interface state machines. The expiration of the Assert timer caused a transition to the NoInfo state, while the expiration of the PruneTimer caused non-root interfaces, that were in a Pruned state, to transition back to the NoInfo state.

  Data packets by being again flooded, recreated the multicast distribution tree by pruning again redundant paths and branches that do not lead to interested receivers.

  With this test, we have verified that the tree only lasted 3 minutes, as specified. Also, after the expiration of this state, we have observed the recreation of the tree due to the flood of data packets through the whole network.

By performing these tests, regarding the interest of hosts, we could verify that the transitions in the Upstream and Downstream Interface state machines were made correctly. Also, the transmitted messages and corresponding state transitions were made within the time intervals specified by the protocol specification, which confirmed that the timers were set correctly.

2.5 State Refresh

Now we will describe the test results regarding the State Refresh mechanism. The same topology of the previous tests was used (Figure 2). All interfaces were set to their default costs (10), except for interface eth2 of R7 that was set to 100. This meant that interface eth0, in all routers, must be set as the root interface for data traffic originated from Source.

In order to perform these tests, all routers enabled the protocol with the State Refresh feature. This caused all routers to include an additional option in the exchanged Hello messages and also to periodically transmit and flood through the whole network a State Refresh message. This information was present in the packet capture of all links.

All corresponding Netkit network configurations, tests and results regarding State Refresh are stored in a branch named "Test_PIM_StateRefresh" at our GitHub repository [10].

Regarding timers, they were set to their default values, which meant that the periodicity of State Refresh messages was 60 seconds (used in the State Refresh timer). Regarding the router directly connected to the source, placed in a Originator state, it will consider the source to be active as long as its Source Active timer does not expire, which was set to 210 seconds whenever it received data packets.

- **Test 16 -** Tree maintenance by State Refresh messages

  All routers should be in a NotOriginator state, except for the router that directly connects the source of multicast traffic, R1, that must be in Originator state. After starting the protocol process in all routers, host Source started transmitting data packets destined to group 224.12.12.12. We executed this test having both receivers interested in multicast traffic of this group.

  In the logs, we have verified that only router R1 transitioned to Originator state regarding this tree. All other routers transitioned to NotOriginator state, by not being directly connected to the source of multicast traffic.

  Then we have continued transmitting data packets and verified that State Refresh messages were being flooded in the network, according to all packet captures. In the switched network, that connects R2, R3, R4, R5 and R6, the first data packet was received at time 49.18 seconds and the first State Refresh message was received at time 109.00 seconds, which was according to the expected periodicity of these messages (60 seconds). Following State Refresh messages also respected this periodicity (at times 169.05, 229.14, 289.15, ...).

We have observed that only the Assert Winner, of a given link, transmitted State Refresh messages. Other routers did not transmit these types of messages, as expected.

Regarding the content of State Refresh messages, these were correct, by being properly dissected and by having their fields set to expected values. Those fields include the RPC of the router that transmitted the message, TTL which was decremented whenever the message was forwarded to a new link and also the flags. Regarding the flags, these included the Prune indicator, Prune Now and Assert Override. The Prune indicator was set according to the Downstream Interface state machine, the Prune Now flag was set on every third transmitted State Refresh message and the Assert Override flag was set according to the Assert state machine.

Regarding the Prune indicator flag, this was visible on links that were pruned out from the broadcast tree, like the link that connects R1 to R2. In this link, after R2 transmitted a Prune message to R1, because the first router lost the Assert in the switched network, this caused following State Refresh messages, originated by R1, to be transmitted with the Prune indicator always set. In the switched network, we have observed that State Refresh messages did not have the Prune indicator flag set due to the interest of both downstream routers, R5 and R6, by reaching interested receivers. Pruned links did not transmit any data packets while the source was active, in a time window of 10 minutes, due to the refresh of the Pruned state, in the Downstream Interface state machine, caused by the reception of State Refresh messages.

Regarding the Assert Override flag of State Refresh messages, in links that elected the Assert Winner with Assert messages, the Assert Winner transmitted these messages with the flag cleared. In links that did not require to elect an Assert Winner with Assert messages, such as the link that connected R1 to R2, because R1 was connected with a non-root interface and R2 was connected to the same link with a root interface, the flag was set. The value of this flag was set accordingly because only routers in a NoInfo state at the Assert state machine, must send these messages with the flag set, like R1 did in the link that connected this router to R2.

Regarding the Assert election, we have just observed the exchange of Assert messages on the arrival of the first data packets. After electing correctly the Assert Winner, through the first Assert messages, following State Refresh messages refreshed the state of the Assert Winner on all routers. This was possible to verify in the logs, by the reception of the State Refresh from the Assert Winner that caused the restart of the Assert timer, used to maintain state regarding the winner of a given link.

During the transmission of data packets, around 10 minutes, R1 remained in the Originator state due to the reception of data packets from Source, causing the reset of its Source Active Timer.

With this test we have verified that State Refresh messages were correctly exchanged by routers. Only the Assert Winner transmitted these messages, as expected.

Regarding the State Refresh state machine, only the router that directly connected to the source of multicast traffic transitioned and remained in the Originator state. All other routers, that were not directly connected to the source, remained in NotOriginator state. By not hearing data packets in pruned links, we have confirmed that these messages correctly refreshed the Assert and Pruned states.

- **Test 17 -** Stop maintaining a tree due to the absence of data and State Refresh messages

  Absence of data packets must cause the router directly connected to the source of multicast traffic, R1, to transition to the NotOriginator state in the State Refresh state machine. State Refresh messages should no longer be transmitted by this router. After Test 16, host Source stopped transmitting data packets.

  We have observed in all links that around 3 additional StateRefresh messages were exchanged after the source stopped transmitting data packets. Then no more State Refresh messages were observed. By looking to the logs, we have verified that this behavior was due to the expiration of the Source Actime timer of router R1, that caused a transition to the NotOriginator state. In this state, the router that directly connects to the source, no longer transmits State Refresh messages.

  In the switched network, after the last transmitted State Refresh message, there were no more messages exchanged regarding this tree.

  In a previously pruned link, like the one that connects R1 to R2, we have observed that after the last State Refresh message was transmitted, Graft and Graft-Ack messages were exchanged. This

was due to the expiration of both the Prune state (in the Downstream Interface state machine of router R2) and the Loser state (in the Assert state machine of router R2). The expiration of both states, caused this router to have a non-root interface included in the OIL at its multicast routing table. This caused router R2 to have its OIL non-empty, triggering the exchange of Graft messages, since the link no longer was considered to be Pruned.

We have observed in the logs, that all routers, at around the same time, timed out the state used to control the Assert and Prune states. This happened around 3 minutes after the last exchanged State Refresh message, as expected.

With this test, we have verified that the router directly connected to the source of multicast traffic controlled its State Refresh state machine, by transitioning to the NotOriginator state when it no longer heard data packets. Also, all routers in the network correctly reacted to the absence of State Refresh messages, by expiring the Assert and Prune states.

We have verified that the router directly connected to the source of multicast traffic reacted correctly to the forwarding and absence of data packets. This regulated the transmission of State Refresh messages that were forwarded throughout the whole network. We have confirmed that these messages were correctly built, by verifying them in *Wireshark*.

All tests proved that each module of the protocol was implemented into accordance to the specification. All timers were set into accordance to the specification, by observing that messages and state transitions were performed in a given time window or after a given amount of time. Control messages were correctly built, being fully dissected by *Wireshark*.

3 HPIM-DM Implementation Test Results

Like PIM-DM, a unicast routing protocol was configured in order for HPIM-DM to correctly determine root and non-root interfaces. For this reason, OSPF [8] was used as the underlying unicast routing protocol, having all routers executing *Quagga* [9], that had an implementation of this protocol.

3.1 Neighborhood maintenance

Regarding the tests of neighborhood relationships, we just needed a single subnetwork with one or more routers connected to it, for this reason we opted to use the topology of Figure 1 that is composed by two routers, R1 and R2. In these tests, we focused only on the formation of neighborhood relationships, without exchanging any tree information with the neighbor router (no trees were previously set). The exchange of tree information during the synchronization will be described on following sections.

All corresponding Netkit network configurations, tests and results regarding establishment of neighborhood relationships, without having trees set up in the network, are stored at our GitHub repository [7] in a branch named "Test_NewProtocol_Sync_Without_Trees".

A synchronization to be successful requires both routers to exchange Sync messages with a bunch of sequence numbers (BootTime, SnapshotSN and SyncSN). In order to accomplish this, both routers must exchange these sequence numbers using a Stop-and-Wait protection, in which there is a Master-Slave election. At the end of this process, both routers need to know the same sequence numbers and a router must have its neighbor router in the SYNCED state.

- **Test 1 -** Establishment of neighborhood relationship between two unknown neighbors, without any existing trees

  In this test, we started first the routing process on R1 and then on R2. We verified that by enabling the process in an interface, this triggered the transmission of an Hello message.

  According to the logs, the first transmitted message by R1 was an Hello message having the BootTime set to 1534977381. Then R2 also transmitted an Hello message with BootTime set to 1534977383. According to the logs, the first transmitted Hello message from R1 was not received by R2 since at that time this router did not have the protocol process started. This explains why the synchronization process only started after the Hello message from R2 was transmitted.

  The first Sync message was transmitted by R1, in response to the Hello message from an unknown neighbor router, R2. Router R1 by being the first one triggering the synchronization process,

considered itself as Master of it, which is possible to verify in the logs by R1 transitioning its neighbor R2 from an UNKNOWN to SLAVE state. The Sync message transmitted by R1 had the BootTimes of both routers set correctly (own BootTime set to 1534977381 and neighbor BootTime set to 1534977383), just like in the Hello messages. Regarding the SnapshotSNs, R1 transmitted a Sync message with its own SnapshotSN set to 1 and the neighbor SnapshotSN set to 0, just like it was expected since R1 has still no information regarding the neighbor router. The SyncSN was set to 0, as expected since this was the first phase of the synchronization process. Regarding the flags, the Master flag was set and the More flag was cleared. These flags were correctly set, since R1 considered itself as the Master of this process and had no more tree information to transmit in following Sync messages.

According to the logs, we verified that R2 heard R1 for the first time due to its Sync message. For this reason, R2 had neighbor R1 transition from an UNKNOWN to MASTER state. In response to the Sync message from R1, R2 transmitted a new Sync message. This message had all SNs correct (own BootTime set to 1534977383, neighbor BootTime set to 1534977381), just like the Hello messages. Regarding the SnapshotSNs, this message had information regarding the neighbor SnapshotSN, which was set to 1, due to the received message and had its own SnapshotSN also set to 1. Regarding the flags, both Master and More were cleared, as expected since the router considered itself as the Slave of the synchronization process and did not have more tree information to be transmitted. The SyncSN was set to 0, since this was the first phase of the synchronization.

R1 by receiving the message from R2, transmitted a new Sync message equal to its first message but now having the SyncSN set to 1. R2 answered back with a message equal to its second message, having 1 also set as the SyncSN. After this, both routers finished successfully the synchronization process, transitioning to SYNCED state, and had exchanged and stored the same SNs.

In this test we have verified that both routers have established a correct neighborhood relationship, having exchanged all information consistently.

- **Test 2 -** Re-establishment of neighborhood relationship after a known neighbor reboots

  After Test 1, we have restarted the protocol process in R2. This caused R2 to transmit an Hello message with a greater BootTime, 1534977413, as expected.

  In the logs, we have verified that R1 by receiving the message from R2, considered its neighbor to have restarted, causing a transition in the neighbor state machine, regarding R2, from SYNCED to SLAVE. This caused R1 to consider itself as the Master of the synchronization process, since it has detected a reboot of a known neighbor due to the reception of a control message with a greater BootTime. This is visible in the packet capture by having R1 transmitting a new Sync message to R2.

  The new Sync message transmitted by R1 had its own BootTime set to 1534977381, like the previous Hello and Sync messages, neighbor BootTime set to 1534977413, due to the received Hello message from R2, own SnapshotSN set to 2, which is greater than the sequence number used in the previous synchronization process as expected, neighbor SnapshotSN set to 0, since this information was still not known, and SyncSN set to 0, due to this Sync message belong to the first phase of the Sync message exchange. The Master flag was set and More flag was cleared, just like the previous synchronization process.

  In response to the message transmitted by R1, R2 considered R1 to be in MASTER state, since the router discovered its neighbor by a Sync message. This caused a new message from R2 to be transmitted to R1, having its own BootTime set to 1534977413, neighbor BootTime set to 1534977381, own SnapshotSN set to 1, neighbor SnapshotSN set to 2 and SyncSN set to 0. The Master and More were cleared, as expected. As we can see, all SNs had consistent information, like the first Sync message transmitted by R1.

  Two more Sync messages were exchanged having all SNs and flags consistent and SyncSN set to 1. At the end, both routers considered each other to be in an SYNCED state, as expected.

  In this test, we have verified that a router can detect when its neighbor has restarted, causing a newer synchronization process to be started. This detection was caused by the greater BootTime obtained after the protocol process restarted on R2.

- **Test 3 -** Neighborhood relationship break after known neighbor fails

After Test 2, we have stopped the protocol process in R2. This caused R2 to no longer transmit Hello messages in the switched network.

After a given amount of time we have verified that in the logs, router R1 stopped considering R2 as a neighbor.

In this test we have verified that the absence of Hello messages from a neighbor router causes a router to break a neighborhood relationship.

3.2 Formation and maintenance of the broadcast tree

Now we will describe the test results regarding the formation of the broadcast tree. In these tests we have verified if the interfaces' roles were selected correctly, if the transmitted control messages were correct and if the state of the tree was set correctly. In this protocol, the formation of the broadcast tree happens due to the exchange of IamUpstream and IamNoLongerUpstream messages. The synchronization between routers, in the formation of neighborhood relationships, did not include the trees of these tests.

All corresponding Netkit network configurations, tests and results regarding the formation of the broadcast tree are stored in a branch named "Test_NewProtocol_BroadcastTree" at our GitHub repository [7].

In these tests and on the following tests, we have enabled the option of "Flooding Initial Data Packets". This meant that when an UPSTREAM router had not received interest information from its neighbors, it considered those neighbors to be interested. This allowed initial data packets to be flooded in order to not be "lost" during the formation of the tree.

- **Test 4 -** Formation of the broadcast tree (root interface selection and tree state)

  In this test, after all routers synchronized with each other, Source transmitted data packets for the first time. The two hosts reported interest in order to verify if data packets were received.

  In the logs, we have verified that all interfaces' roles were correctly selected, i.e. eth0 was selected as the root interface in all routers, except for router R7 that selected eth2 as its root interface. All the remaining interfaces were selected as non-root type. It was also possible to verify that the tree state regarding (10.1.1.100, 224.12.12.12) transitioned from INACTIVE to ACTIVE state in all routers. In some routers, prior to the transition to ACTIVE state, there was a transition to UNSURE state. This meant that all routers received IamUpstream messages through their root interface, but some routers received first IamUpstream messages through their non-root interfaces.

  We can observe that in the following links there were IamUpstream messages exchanged:

  - Links that connects R1 to R2, R1 to R3 and R1 to R4: Only R1 transmitted an IamUpstream(S,G) message.

    This was expected since only R1 was connected by a non-root interface to those links. All messages transmitted by R1 had an RPC of 0 because this router was directly connected to the source of multicast traffic.

  - Link that connects R2, R3, R4, R5 and R6: Only R2, R3 and R4 transmitted IamUpstream(S,G) messages.

    This was expected since only those routers were connected by non-root interfaces to that link. These routers transmitted IamUpstream(S,G) messages with the same RPC of 20. This RPC was expected because all routers set their interfaces' cost to 10.

    We have observed in the packet capture that the first IamUpstream message was transmitted by R2. According to the logs, this caused routers R5 and R6 to transition to ACTIVE state due to the presence of an UPSTREAM neighbor connected to their root interfaces. In the case of routers R3 and R4, these transitioned to UNSURE state, suggesting that these routers received their first IamUpstream message through their non-root interfaces.

    Then according to the logs, R3 and R4 transitioned to ACTIVE state due to the reception of R1's message. This caused the transmission of IamUpstream messages from both R3 and R4.

  - Link that connects R2 to R7: Only R2 transmitted an IamUpstream(S,G) message.

    This was expected because only R2 was connected by a non-root interface to that link. That message was transmitted with an RPC of 20, as expected. This IamUpstream message by being received through R7's root interface caused a transition to ACTIVE state in this router.

– Link that connects R5 to R7: both routers transmitted IamUpstream(S,G) messages.

  Since both routers were connected to each other by non-root interfaces, the transition to ACTIVE state caused the transmission of IamUpstream(S,G) messages from both routers. These messages were transmitted with RPC of 30, as expected.

As expected, root interfaces did not transmit any IamUpstream messages. They only transmitted interest messages, such as Interest or NoInterest. These interest messages will be further analyzed in the next tests.

We have observed in all packet captures that after an IamUpstream message was received, all neighbors acknowledged that message with an ACK message. Those ACK messages were correct by having information from the IamUpstream message, such as a reference of the tree (source IP and group IP) and its sequence number. Beside this information, they also included information regarding the synchronization process, such as BootTimes and SnapshotSNs. These sequence numbers will be further analyzed in the next tests. Also, we did not observe any retransmissions of IamUpstream messages, which confirmed that the ACKs were correctly received and had consistent information to properly acknowledge those IamUpstream messages.

During the formation of the tree, both members received all data packets regarding this tree, which confirmed that the protocol was flooding data packets during the formation of the broadcast tree, as expected.

With this test we have verified that the broadcast tree was correctly built by selecting interfaces' roles according to the unicast routing table (root and non-root interfaces). We have confirmed that all routers ended up in ACTIVE state, in the Tree state machine, due to the exchange of IamUpstream messages, which were correctly acknowledged.

- **Test 5 -** Broadcast tree reconfiguration in reaction to RPC changes detected by unicast routing protocol

  After Test 4, we have changed the cost of interface eth2 of R7 to 100 in order to change its role. This change should cause interface eth0 of R7 to be selected as the new root interface.

  By analyzing the logs we can confirm this expected behavior. Also, the tree state in this router remained in ACTIVE state due to the presence of an UPSTREAM neighbor connected to the new root interface (R2).

  We have verified that interface eth2 of R7, the new non-root interface that was previously root type, transmitted an IamUpstream message. The transmitted IamUpstream included an RPC of 40, as expected. Also, interface eth0, the new root interface that was previously non-root type, transmitted an IamNoLongerUpstream message and then an interest message destined to R5 (router responsible for forwarding data packets in that link). These transmitted messages were expected according to the protocol specification.

  Then data packets were transmitted and these were received by both members, without any duplications, through the reshaped tree. This test proved that the protocol reacted correctly to an RPC change that resulted in the modification of interfaces' roles.

- **Test 6 -** Broadcast tree reconfiguration in reaction to router failure, detected by both the unicast routing protocol and the multicast routing protocol

  After Test 5, we opted to shutdown interface eth0 of router R5. This should cause a change of the interfaces' roles at router R7, in order to have eth2 as its new root interface.

  In this test we started by disabling the protocol process in interface eth0 of R5, which simulated that the multicast routing protocol reacted first to this change. Since this interface was root type, this caused the protocol to no longer consider to be connected by UPSTREAM neighbors to that interface, causing router R5 to consider the tree to be in INACTIVE state and transmit an IamNoLongerUpstream message through its non-root interface, eth1, which was received by router R7's root interface (eth0). Since R5 was the only UPSTREAM neighbor connected to the root interface of R7, this caused a Tree state transition from ACTIVE to UNSURE in R7. The transition to UNSURE was due to the presence of UPSTREAM neighbors connected to interface eth2 of R7, a non-root interface. This transition caused the transmission of an IamNoLongerUpstream through R7's non-root interface, eth2, as observed in the packet capture of that link.

Then we have shutdown interface eth0 of R5, causing the unicast routing protocol to detect and react to this change. This caused a change in the unicast routing table of R7, modifying the interfaces' roles of this router, being eth2 selected as its new root interface. This change in the interfaces' roles caused a transition in the Tree state from UNSURE to ACTIVE at R7, as observed in the logs, due to the presence of an UPSTREAM neighbor connected to its new root interface (R2). The new non-root interface, eth0, in reaction to this Tree state transition transmitted an IamUpstream message, as expected. The RPC present in this last message was of 120, as expected due to the interface cost of eth2 being 100 (in Test 2).

Following data packets transmitted by Source were correctly forwarded and received by both hosts.

In this test it was possible to verify that all control messages were correctly transmitted in reaction to transitions in the Tree state machine. These caused the transmission of IamUpstream and IamNoLongerUpstream messages depending on the state transition and on the interface type. All messages were correctly acknowledged, like the previous tests.

According to these tests, the tree was correctly formed even in the presence of RPC changes and router failures that caused a modification of the interfaces' roles. The tree state was set according to the presence of UPSTREAM neighbors in all interfaces, like the protocol specified. All messages were correctly transmitted based on those state transitions and were correctly acknowledged by all neighbors connected to the same link.

3.3 Assert mechanism

Now we will describe the test results regarding the Assert mechanism. These tests were similar to the ones that were performed on the PIM-DM implementation, but now we did not require to retransmit data packets in order to cause a new reelection since whenever the current Assert Winner suffered a change, previously stored state regarding UPSTREAM neighbors was used in the new reelection. We used the same network topology of previous tests, represented by Figure 2. In these tests we only focused on what happened in the switched network, that connected routers R2, R3, R4, R5 and R6.

All corresponding Netkit network configurations, tests and results regarding the election of the router responsible for forwarding multicast traffic are stored in a branch named "Test_NewProtocol_Assert" at our GitHub repository [7].

The same configuration of the previous tests, in the unicast routing protocol, was made in order to have all routers knowing about all subnetworks. Each interface was set with a cost of 10. Also, we only started these tests after all routers established a neighborhood relationship and correctly synchronized with each other. The tree that was used on these tests was not previously formed in order to test the Assert mechanism triggered only by IamUpstream messages. As in the case of PIM-DM, by having all UPSTREAM routers (R2, R3 and R4) offering the same RPC to the Source subnet, 20, the IP address was used in this election (greater IP wins the Assert).

- **Test 7 -** Election of the AW in shared links

  Both hosts initially reported interest in order to verify if initial data packets were received. After that, the Source started transmitting data packets.

  In the switched network, according to the packet capture, router R2 was the first one transmitting an IamUpstream message. This was followed by R4 and then by R3. These transmissions were caused by these routers having an UPSTREAM neighbor, router R1, connected to their root interfaces.

  According to the logs, router R2 was the first one that transitioned to an ACTIVE state in its Tree state machine. This explains why this router was the first one transmitting an IamUpstream message. Also, according to the logs routers R3 and R4 transitioned first from an INACTIVE state to an UNSURE state in their Tree state machines. This was due to the first UPSTREAM neighbor being discovered by a non-root interface (R2's message). Routers R5 and R6 transitioned immediately from INACTIVE to ACTIVE state in their Tree state machines, due to the IamUpstream message, from R2, being received on their root interfaces.

  Then we verified that router R4 transitioned to ACTIVE state, due to the IamUpstream message from R1 being received in its root interface. Then the same happened to router R3. The transition to ACTIVE state caused both routers to transmit IamUpstream messages to the switched network, by their non-root interfaces.

Regarding the Assert state machine, according to the logs, all routers started initially in ASSERT WINNER state. When R2 transmitted its IamUpstream message, both R3 and R4 transitioned to ASSERT LOSER state, as expected, due to their Tree state machines being in UNSURE state.

We have verified that when R4 transitioned to ACTIVE state, in its Tree state machine, this caused a transition to ASSERT WINNER state in interface eth1 of R4, in its Assert state machine, due to the presence of an UPSTREAM neighbor connected with a worse "Assert state" than itself (R2). This message by being received by R2 and R3, caused both routers to transition to ASSERT LOSER in their Assert state machines.

We have also verified in the packet capture that after the exchange of IamUpstream messages, both R5 and R6, transmitted their interest messages destined to R4 (destination IP address was unicast). This proved that both routers elected correctly R4 as the new ASSERT WINNER, since interest messages are always destined to the router that wins the Assert.

As the previous tests, all messages were acknowledged correctly. IamUpstream messages were acknowledged by all routers since they were destined to all routers (destination IP address was multicast) and interest messages were only acknowledged by a single router (destination IP address was unicast). The ACK messages had all correct information such as Sequence Numbers and tree identification. Regarding IamUpstream messages, all routers transmitted the same RPC, 20, as expected.

Client2 received initially a triplication of the first data packets, as expected, due to those packets being initially forwarded by R2, R3 and R4. Then following data packets were received without duplications, due to having a single Assert Winner connected to the switched network.

In this test we have verified that all routers elected correctly the Assert Winner by exchanging IamUpstream messages. These messages were correct, by being correctly dissected and by being correctly acknowledged. Downstream routers by sending their interest information destined to R4, proved that those routers elected that router as the new Assert Winner.

- **Test 8 -** Reelection of the AW in a shared link, in case the previous one fails

  After Test 7, we stopped the protocol process in interface eth1 of R4. This interface was considered to be the Assert Winner of the switched network. By stopping the protocol process in that interface, just like in PIM-DM, a new Hello message was transmitted, informing all routers that this interface was shutdown. This allowed all routers to react quickly to its removal.

  We could verify in the packet capture, that at time 83.007 seconds an Hello message from R4 was transmitted, having the option Hello Holdtime set to 0. The reception of this message caused all routers to stop considering R4 as a neighbor.

  According to the logs, router R3 transitioned from ASSERT LOSER to ASSERT WINNER state, as expected. Router R2 remained in ASSERT LOSER state, due to the presence of an UPSTREAM neighbor with a better Assert state, R3.

  In the packet capture we can verify that right after the referred Hello message was transmitted, at time 83.037 and 83.041 seconds, interest messages from R5 and R6 were transmitted to the new Assert Winner (R3). This proved that both downstream routers elected correctly R3 as the new Assert Winner.

  The Source by transmitting data packets, these were received by both hosts. The reception in Client2 was due to these being forwarded by the new Assert Winner (R3).

  In this test we have confirmed that the removal of a router that was considered to be the Assert Winner of a given link, caused all routers to recalculate the new Assert Winner based on the presence of UPSTREAM information that was previously stored. IamUpstream messages were not exchanged proving that this election took into account only the information that was previously stored.

- **Test 9 -** Reelection of the AW in a shared link, in case the previous one changes its interface role (its interface becomes root type)

  After Test 8, interface eth1 of R3 was elected as the new Assert Winner of the switched network. In this test, like it was performed in the same test of PIM-DM, we changed the cost of interface eth0 of R3 to 100, in order for eth1 of the same router to become root type.

After changing the cost of R3's interface, we have verified in the logs that router R3 had its interfaces' roles changed. Interface eth1 became root type. The change of the interfaces' roles in R3, caused it to transmit an IamNoLongerUpstream message though its new root interface, eth1, as it was observed in the packet capture.

This packet by being received by R2 caused this router to consider itself as the new Assert Winner of the switched network, which was observed in the logs, by the transition to ASSERT WINNER state in its Assert state machine. This happened because interface eth1 of R2 did not have any information regarding UPSTREAM neighbors connected to that interface.

Also in the packet capture we can observe that after the transmission of the IamNoLongerUpstream, from R3, all downstream routers transmitted interest messages destined to R2, which confirms that all routers considered that router as the new Assert Winner of the switched network. These interest messages were transmitted by R3, R5 and R6, via their root interfaces.

Source by transmitting data packets, these were received by both hosts. The reception of data packets in Client2 proved that the new Assert Winner, R2, correctly forwarded them.

All exchanged messages, such as IamNoLongerUpstream and interest messages were correctly acknowledged, just like in the previous tests.

As it was verified in this test, a non-root interface of an Assert Winner by becoming root type, caused the new root interface to transmit an IamNoLongerUpstream, in order for all routers to not consider it as UPSTREAM. This triggered a reelection of the new Assert Winner, based on the information that was already stored. There were no additional IamUpstream messages transmitted, which proves that only previously stored information was used in the election. All routers by transmitting interest messages destined to the new Assert Winner, R2, proved that this election was performed correctly.

Like in the tests of PIM-DM, all tests elected the same Assert Winners, since this election took into account the same type of information, RPC and IP address. We had the same results, differing only that in this protocol this election was not performed by the exchange of data packets, i.e. the reception of data packets in non-root interfaces did not trigger the election, instead already stored information was used. All exchanged control messages and all state transitions in both the Tree and Assert state machines were expected, in accordance to the protocol specification.

3.4 Forwarding based on Interest information

Now we will describe the tests regarding the decision of data forwarding based on the interest of NOT UPSTREAM neighbors. These tests were similar to the ones that were performed on the PIM-DM implementation, regarding the Prune, Join and Graft messages. Like previous tests, we used the same network topology, represented by Figure 2. In these tests we focused on the exchanged control messages of all routers and also on the decision of the Assert Winner of all links regarding the interest of NOT UPSTREAM neighbors. In order to detect this decision, we logged all events of non-root interfaces, regarding the global decision of interest, i.e. there is downstream interest if at least one neighbor is interested or no downstream interest otherwise.

All corresponding Netkit network configurations, tests and results regarding the interest decision of the router responsible for forwarding multicast traffic are stored in a branch named "Test_NewProtocol_Interest" at our GitHub repository [7].

Regarding the unicast routing protocol, we changed the cost of interface eth2 of router R7 to 100, in order for eth0 of this same router be selected as root type. This allowed the interest decision on the switched network to be controlled by the membership of both hosts.

We performed these tests after all neighbors correctly formed neighborhood relationships with each other. The tree formation was done only by the exchange of IamUpstream messages. In these tests we did not focus on the synchronization process.

- **Test 10 -** Correct multicast forwarding decision based on neighbors' interests, after the tree is initially built

  We started initially by having both hosts interested in receiving multicast traffic of group 224.12.12.12. Then Source started transmitting data packets and all routers formed the tree.

Since all routers had the option "Flood Initial Data Packets" enabled, we verify in the logs that all interfaces, that connected to at least one neighbor, started initially by considering that there was interest by downstream routers. This allowed to flood initial data packets through the whole network, being received by both hosts. Interfaces eth1 of R7 and eth1 of R6 did not consider that there was interest from downstream neighbors since they were not connected to other routers.

With the exchange of IamUpstream messages due to the tree formation we observed the following in each link:

- Link that connects R1 to R2:

  According to the packet capture, R2 started by informing R1 that it was interested, via an Interest message. This is explained by this router initially consider itself as the Assert Winner of the switched link and of the link that connects R2 to R7 and by considering that there was downstream interest in both links (due to the enabled option).

  Then we verified, with the election of the Assert Winner in the switched link and by the absence of NOT UPSTREAM neighbors connected to the link R2-R7, this router transmitted a NoInterest message to R1.

  R1 then no longer considered the link that connected this router to R2 to have downstream interest, as expected.

- Link that connects R1 to R3:

  We verified that R3 started by informing R1 that it was not interested. This can be explained by R3 consider itself initially as the Assert Loser of the switched link, since R2 transmitted first an IamUpstream message when R3 was considered to be in INACTIVE state in its Tree state machine, causing a transition to UNSURE state.

  Since R3 was in UNSURE state and there was an UPSTREAM neighbor connected to its non-root interface, its interface was not part of the broadcast tree (it was pruned). By not having any interface that would forward data packets, when R3 detected the existence of an UPSTREAM neighbor connected to its root interface, it transmitted initially a NoInterest message. Then R3 by transitioning to ACTIVE state and consider itself as the Assert Winner of the switched link, because at that time only R2 had transmitted the IamUpstream message and R3 had a better Assert state, this router transmitted an Interest message.

  Then R3 considered again to be the Assert Loser of the switched link, due to presence of R4, causing the transmission of a NoInterest message to R1. From this point on there were no more interest messages exchanged between both routers. At the end, interface eth2 of R1 considered this link to not be connected to interested downstream routers, causing the pruning of this interface from the distribution tree.

- Link that connects R1 to R4:

  The same thing that was described in link R1-R3 happened in this link. The only difference was that the final message transmitted by R4 to R1 was an Interest message, due to this router consider itself as the Assert Winner of the switched network and by being connected to interested downstream neighbors.

- Link that connects R2, R3, R4, R5 and R6:

  We have observed that the first router that transmitted an IamUpstream message was R2, then R4 and finally R3.

  When R2 transmitted its message, all routers transmitted interest messages to this router, including R3 and R4. These two routers transmitted NoInterest messages to R2 since they were connected by non-root interfaces to the switch. The other routers transmitted Interest messages, as expected.

  Then, when R4 transmitted its IamUpstream message, only R5 and R6 transmitted Interest messages to it, since it was the new Assert Winner of the link. R3 did not transmit an interest message to R4 because at that time it already considered the tree to be in ACTIVE state.

  Finally, when R3 transmitted its IamUpstream message, there were no interest messages exchanged. This was expected because all routers already were storing at that time information regarding the RPC of R4. When R3 transmitted its IamUpstream message, R4 was already considered to be the Assert Winner of the switched network.

The reception of Interest messages at R4 caused this router to consider the switched network to have interested downstream neighbors, as expected. The conclusion reached by R2 and R3, regarding the existence of interested downstream neighbors, was not relevant since only the Assert Winner of a link must have a consistent view regarding this matter. NOT UPSTREAM neighbors only send their interest information to the current Assert Winner, so Assert Losers do not require to have that same view since they do not make forwarding decisions in this link. If an Assert Loser becomes Assert Winner, all NOT UPSTREAM neighbors would retransmit their interest to the new winner.

- Link that connects R5 to R7:

  When R5 transmitted its IamUpstream message, R7 sent an Interest message to it. There were no more interest messages exchanged, since Client1 remained interested in multicast data packets.

  The reception of the Interest message by R5, caused this router to still consider this link to have interested downstream routers.

- Link that connects R2 to R7:

  Both routers were connected to this link by non-root interfaces. The exchange of IamUpsteam messages by both routers caused these two routers to consider that there was no downstream interest in this link. Interest/NoInterest messages were not exchanged, but both routers reached this conclusion correctly, because they both considered to be UPSTREAM regarding this tree. Since the link only connected UPSTREAM neighbors, they reached to this conclusion without having interest messages involved.

All links reached the right conclusion regarding the existence of interest from downstream neighbors. This happened due to the exchange of IamUpstream and Interest/NoInterest messages.

Like previous tests, all messages were correctly interpreted and acknowledged, which explains the transmission of interest messages destined to the Assert Winner of each link.

- **Test 11 -** Correct multicast forwarding decision after a neighbor changes its interest (a neighbor becomes not interested but there are still other neighbors interested)

  After Test 10, we closed the socket used to receive data packets at Client2. This caused the transmission of IGMP Leave, being received by R6.

  Since interface eth1 of R6 was not connected to any router and the state of group 224.12.12.12 at IGMP was in a state in which it indicated that there was no interest from hosts, this caused interface eth0 of R6, the root interface, to transmit to the switched link a NoInterest message destined to R4.

  We have observed that this message was received by R4, which answered back with an ACK message. We have verified in the logs that R4 still considered that there was interest from NOT UPSTREAM neighbors, due to this router having stored state regarding the Interest message that was transmitted in the previous test, from router R5. Router R5 did not transmit a new interest message to R4, as expected.

  We have then transmitted data packets and these were correctly received by Client1 that was still interested. These packets appear on the packet capture.

  With this test, we have verified that R4 still considered that there was interest from downstream neighbors, due to the storage of interest information. The NoInterest message did not cause the switched link to be removed from the tree, due to the existence of an INTERESTED neighbor (R5).

- **Test 12 -** Correct multicast forwarding decision after a neighbor changes its interest (a neighbor becomes not interested and there is no interested neighbor)

  After Test 11, we closed the socket from the remaining interested host. This triggered the transmission of an IGMP Leave, from Client1, causing interface eth1 of R7 to stop considering to be connected to interested hosts.

  We have verified in the packet capture that R7 transmitted to R5 a NoInterest message, which was properly acknowledged. In the logs we can verify that interface eth1 of R5 stopped considering to be connected to interested downstream neighbors upon the reception of this message. This is explained by R5 being connected to only one neighbor, in that interface, that was no longer

interested in receiving data packets from this tree. This caused the transmission of a NoInterest message from interface eth0 of R5, its root interface, destined to R4.

We observed that the reception of the NoInterest message from R5, caused R4 to no longer consider that there was interest from downstream neighbors. This is explained by R4 not storing information regarding interested neighbors. At that time, R4 considered R2 and R3 to be UPSTREAM and R5 and R6 to be NOT INTERESTED. Since it was not connected to INTERESTED neighbors, this caused the Assert Winner of the switched network to prune its non-root interface. R4 by having its only non-root interface pruned, caused the transmission of a NoInterest message by its root interface, eth0, in order to prune the distribution tree.

The message received by R1 was properly acknowledged. Then Source transmitted data packets and these were not forwarded to any link.

In this test, we proved that the forwarding decision was made taken into account the state that was already stored, regarding all neighbors. When a router no longer connected to INTERESTED neighbors, this caused the prune of the tree.

- **Test 13 -** Correct multicast forwarding decision after a neighbor changes its interest (all neighbors were not interested and one them becomes interested)

  After Test 12, we opened the socket at Client2. This caused the IGMP implementation of Client2 to inform router R6 that there was interest from an host connected to that link, via an IGMP Report message.

  In the packet capture we can verify that router R6 transmitted an Interest message destined to R4. In the logs, after the reception of this message by R4, this router considers that there is again downstream interested neighbors. This caused interface eth1 to belong again to the OIL of this tree. R4 then transmitted an Interest message through its root interface, eth0, in order to inform about this change to R1.

  In the packet capture we have verified that R4 transmitted an Interest message to R1, which was properly acknowledged. In the logs, we have observed that this message caused R1 to consider again that there was interest from downstream neighbors regarding this tree.

  By having the Source transmitting again data packets, these were correctly received by Client2, proving that the tree has been reshaped in order to attach the new member to it.

  In this test we have verified that the tree reacted to the interest of the new host. This caused the exchange of Interest messages in order for data packets from Source to be forwarded to the new member.

In these tests we have confirmed that the implementation reacted correctly to all events regarding interest information. The tree was "pruned" when hosts manifested their disinterest and was "grafted" when they manifested interest in those same data packets. This behavior was caused by the exchange of IamUpstream, Interest and NoInterest messages, causing all routers to store information regarding their neighbors. Just like the previous tests, all messages were correctly acknowledged.

3.5 Discovery of tree information based on synchronization

We will now describe the tests regarding the exchange of tree information using Sync messages. This can happen whenever a new neighborhood relationship is formed and there were already trees set up in the network. In order to test this, we will reuse the same topology of previous tests (Figure 2). In these tests we will start the protocol process in all routers, then form multiple trees and finally restart a router (considered as UPSTREAM and NOT UPSTREAM for those same trees). Multiple trees will be formed in order to verify if the fragmentation of this information is being done correctly. In order to do this, we have manually configured all routers to only include information about 5 trees per Sync message, in order to test the fragmentation with a lower number of trees per message, instead of fragmenting Sync messages using MTU information.

All corresponding Netkit network configurations, tests and results regarding the synchronization of trees between routers are stored in a branch named "Test_NewProtocol_Sync_With_Trees" at our GitHub repository [7].

Before we started these tests, all routers synchronized with each other when there were no trees configured on routers. Then Source transmitted data packets to 20 groups (from 224.12.12.12 to 224.12.12.32).

All routers formed these trees with the exchange of IamUpstream messages. Then we focused on what happened when an UPSTREAM and NOT UPSTREAM routers rebooted. In these tests we focused on the switched network, being R4 the UPSTREAM router that would reboot and R5 the NOT UPSTREAM router that would reboot.

- **Test 14 -** Tree formation triggered by the synchronization process with a known neighbor (reboot of an UPSTREAM neighbor)

  After all trees were set correctly, we have rebooted the router elected as Assert Winner in the switched network, R4. This was performed by removing interface eth1 from the protocol process and then re-enable this interface. Since the root interface of R4 was still being monitored by the protocol process, all trees remained stored in R4, since the root interface was connected to an UPSTREAM neighbor (R1).

  After re-enabling the interface in R4, we have verified in the packet capture and in the logs that this interface formed correct neighborhood relationships with all neighbors (R2, R3, R5 and R6), by exchanging Sync messages with them. We have verified that R5 and R6 did not include any tree in their transmitted Sync messages, since these routers were connected by a root interface to the link that connected them to the rebooted router, i.e. they were not considered to be UPSTREAM. Regarding the remaining routers, R2, R3 and R4 have included all the same trees in Sync messages (from 224.12.12.12 till 224.12.12.32).

  Regarding the synchronization process, all neighborhood relationships were formed with Sync messages ending at SyncSN of 5. This was expected since there were 20 trees to be exchanged, each Sync message only included 5 trees and an additional Sync message was required to terminate the synchronization process (when both routers transmitted Sync message with More flag cleared).

  Regarding the sequence numbers, we have verified that these were consistent in all exchanged Sync messages between each pair of neighbors. Also, all SnapshotSNs of routers were greater than the SNs used to transmit messages prior to this synchronization, in IamUpstream and in Interest/NoInterest messages. For example, the last transmitted message by R5 was a NoInterest message regarding tree (10.1.1.100, 224.12.12.17) with SN=107 and the SnapshotSN of this same router used to synchronize state with R4 was 108.

  After the Sync process has finished, we have verified that all NOT UPSTREAM routers, R5 and R6, transmitted interest messages to R4. This was expected because R4 was considered the new Assert Winner and whenever there is a change of the router responsible for forwarding data packets to a given link, routers need to manifest their interest to it. These interest messages were acknowledged by R4, with ACK messages that included all SNs that were exchanged in the synchronization process. Example, R4 and R5 have synchronized with the following SNs: R4's BootTime=1535061072, R4's SnapshotSN=1, R5's BootTime=1535060980, R5's SnapshotSN=108, which was included in all ACKs transmitted by R4 to R5 in response to R5's interest messages. Routers that were UPSTREAM did not transmit additional messages after the synchronization finished, as expected.

  With this test we have verified that routers were able to exchange tree information whenever an existing neighbor restarted. All tree information was included in Sync messages and interest information was transmitted after the synchronization was successful.

- **Test 15 -** Tree formation triggered by the synchronization process with a known neighbor (reboot of a NOT UPSTREAM neighbor)

  A NOT UPSTREAM router by rebooting must send interest information regarding all trees discovered during the Sync process. In this test, we have restarted the protocol process in interface eth0 of R5. When the interface was enabled, all routers triggered a new synchronization with it.

  Only R2, R3 and R4 included information regarding all existing trees, like the previous test, since these routers were the only ones connected by non-root interfaces to the link, i.e. were the only routers that were considered as UPSTREAM. Routers R5 and R6 exchanged Sync messages with no trees, because they were both connected by root interfaces to the link, i.e. they were considered as NOT UPSTREAM.

  After the synchronization finished, router R5 transmitted interest messages destined to R4, which was the router elected as the Assert Winner of the link. These messages were then acknowledged by R4, with all information obtained during the synchronization (SNs), just like the previous tests.

In this test we have confirmed that only UPSTREAM routers included tree information in Sync messages. After a NOT UPSTREAM router finished synchronizing with all routers, it transmitted interest messages to the router that was elected as the Assert Winner of each exchanged tree.

3.6 Tree maintenance in the presence of loops

All described tests up until now have tested the maintenance of a tree in topologies that do not have loops. While the topology of Figure 2 might look like it has a loop, in the connection of routers R2, R7 and R5, this is not true, since the selection of the root interface can by itself break the loop, since router R2's root interface is eth0.

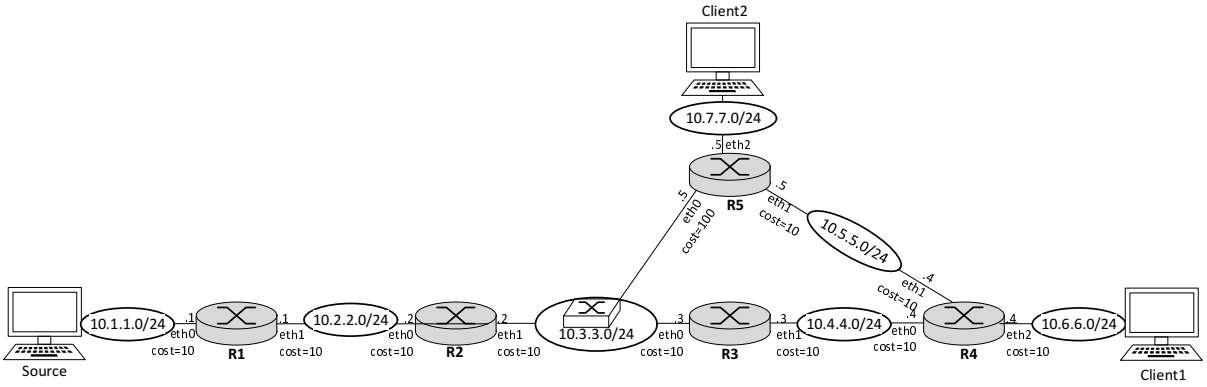

Figure 3: Topology used to test the maintenance of a tree in the presence of loops.

In order to test the maintenance of tree state in the presence of loops, we used a new topology, illustrated by Figure 3. In this figure, near each interface it is detailed its cost, which would be used by the unicast routing protocol to calculate the RPC to every subnet. This information would be obtained by the multicast routing protocol in order to determine the root interface and also the RPC to the source of multicast traffic.

In Figure 3, router R1 is the originator because it is directly connected to the Source, being all the remaining routers considered as non-originators. Interface eth0 is the root interface in routers R1, R2, R3 and R4, while interface eth1 is the root interface in R5, since these interfaces offer the lowest RPC, in each router, to the source subnet.

As we can see, routers R3, R4 and R5 can form a loop, since the root interface of R3 is connected to the non-root interface of R5, the root interface of R5 is connected to the non-root interface of R4 and the root interface of R4 is connected to the non-root interface of R3. For this reason, a router may falsely determine that a tree is ACTIVE, due to these routers informing each other that they are UPSTREAM (maintenance of state due to a loop).

In order to not maintain a tree indefinitely, a router uses the feasibility condition in order to determine a tree state. In order for a tree to become ACTIVE, the feasibility condition must hold, for this reason the RPC is used in the calculation of a tree state. In these tests we will verify if when the source transmits data packets, all routers can determine that the tree is in ACTIVE state. Then if the source stops transmitting data packets, if all routers can correctly determine that the tree is in INACTIVE state.

All corresponding Netkit network configurations, tests and results regarding the maintenance of tree state in the presence of network loops are stored in a branch named "Test_NewProtocol_Loop" at our GitHub repository [7].

- **Test 16 -** Tree formation in the presence of network loops

  Data packets originated by the originator must trigger the formation of the tree. All routers, eventually will be in an ACTIVE state regarding this (S,G) tree.

  We started by initiating the multicast routing protocol in all routers and then the Source started transmitting multicast data packets destined to group G (224.12.12.12). Regarding the hosts, both have reported interest in receiving multicast traffic regarding group G.

  According to the logs all routers reached an ACTIVE state regarding this tree, which confirms that the tree was correctly built.

Regarding the exchange of messages:

- Link R1-R2:

  Router R1 started by transmitting an IamUpstream(S,G) with RPC set to 0. Then an Interest message, from R2, was transmitted to R1. Both messages were correctly acknowledged.

  According to the logs, the reception of the IamUpstream message, by R2, causes this router to consider the tree to be in ACTIVE state.

- Switched network:

  Router R2 started by transmitting an IamUpstream(S,G) with RPC set to 20.

  According to the logs, the reception of this message by R3 causes this router to consider the tree to be in ACTIVE state since it was received in its root interface. The reception of the same message by R5 causes this router to consider the tree to be in UNSURE state since it was received in a non-root interface.

  Then both routers transmitted interest messages destined to R2, the Assert Winner. R3 transmitted an Interest message, since there was downstream interest regarding this tree, while R4 transmitted a NoInterest message, because it was connected to the link by a non-root interface.

  Then router R4 transmitted an IamUpstream(S,G) message with RPC set to 50. This was due to the propagation of Upstream information downwards in the tree (from R3 to R4 and then from R4 to R5), causing R4 to consider the tree to be in ACTIVE state.

  No more messages were exchanged. All described messages were correctly acknowledged.

- Link R3-R4:

  Router R3 by considering the tree to be ACTIVE transmitted an IamUpstream(S,G) with RPC set to 30.

  The reception of this message by R4, according to the logs, caused this router to consider the tree to be ACTIVE. Also in reaction to the reception of this message, R4 transmitted an Interest message destined to R3.

  No more messages were exchanged and all described messages were correctly acknowledged.

- Link R4-R5:

  The transition to ACTIVE state by router R4, caused this router to transmit an IamUpstream(S,G) with RPC set to 40.

  This message was received by R5, which caused the already maintained tree to transition from UNSURE to ACTIVE state. In reaction to this transition, router R5 transmitted an Interest message destined to R4.

  Just like in all links, no more messages were exchanged and all described messages were correctly acknowledged.

As we can verify, all routers reached an ACTIVE state regarding the (S,G) tree. This was due to the presence of UPSTREAM neighbors connected to a router's root interface and also by respecting feasibility condition.

This condition was respected in all links:

- Link R1-R2: RPC advertised by R1 (0) is lower than the RPC of R2 (20);
- Switched network: RPC advertised by R2 (20) is lower than the RPC of R3 (30). Since this RPC is better than the one advertised by R5, the latter is not checked;
- R3-R4: RPC advertised by R3 (30) is lower than the RPC of R4 (40);
- R4-R5: RPC advertised by R4 (40) is lower than the RPC of R5 (50);

As it was verified, all routers reached to the correct state regarding the (S,G) tree.

- **Test 17 -** Tree removal in the presence of network loops

  Absence of data packets must trigger the removal of a tree. All routers must eventually transition to INACTIVE state regarding (S,G) tree. By having the Source stopped transmitting data packets, this caused the originator to eventually trigger the removal of state regarding it, in all routers. This happened around 213 seconds after the tree was initially formed.

According to the logs, the exchange of IamNoLongerUpstream messages, caused all routers to eventually consider the tree to be in INACTIVE state, as expected. This confirmed that all routers eventually stopped having UPSTREAM neighbors connected to any interface, causing the removal of the tree.

The following happened in each link:

- Link R1-R2:

  The first router to transition to INACTIVE state was R1. This transition was performed on its own, due to the expiration of its Source Active timer.

  The transition of R1's tree state caused this router to transmit an IamNoLongerUpstream(S,G) by its non-root interface.

  This message was received by R2, causing a transition from ACTIVE to UNSURE state, according to the logs. This was accomplished since R2, at that point in time, was still connected to an UPSTREAM neighbor by its non-root interface (R5).

  R2 eventually transitioned to INACTIVE state, in reaction to the absence of UPSTREAM neighbors connected to its non-root interface.

  Only the IamNoLongerUpstream(S,G) message was transmitted, being acknowledged by R2. No more messages were exchanged.

- Switched network:

  The initial transition from ACTIVE to UNSURE state, at R2, caused a transmission of an IamNoLongerUpstream(S,G) message.

  The reception of this message caused all routers to consider that the Assert Winner of this link was router R5 since it was the only UPSTREAM neighbor. Router R3 by recalculating its tree state, transitioned to UNSURE state, due to the violation of the feasibility condition, by having the RPC advertised by R5 (50) not lower than R3's own RPC (30). This caused R3 to correctly suspect about the existence of a loop, causing a transition to UNSURE state.

  By having the Assert Winner changed, this caused new interest messages to be transmitted to R5.

  Router R3 transmitted an Interest message since it was still connected to downstream routers interested in receiving traffic from this tree.

  Router R2 transmitted a NoInterest message because its interface was non-root type.

  Eventually R5 transmitted an IamNoLongerUpstream(S,G) causing all routers in this link to no longer be connected to UPSTREAM neighbors. In reaction to this message, all routers connected to this link, transitioned the tree to INACTIVE state.

  Like the previous link, all messages were correctly acknowledged.

- Link R3-R4:

  Router R3 by transitioning to UNSURE state, caused a transmission of an IamNoLongerUpstream(S,G) message, which was correctly acknowledged by R4.

  According to the logs, the reception of this message by R4, caused the tree in this router to transition directly from ACTIVE to INACTIVE state.

  No more messages were exchanged in this link.

- Link R4-R5:

  The transition to INACTIVE state in R4 caused a transmission of an IamNoLongerUpstream(S,G) message, which was properly acknowledged by R5.

  According to the logs, the reception of this message by R5 caused router R5 to transition from ACTIVE to INACTIVE state, due to the absence of UPSTREAM neighbors connected to this router.

  No more messages were exchanged.

As we have verified, the feasibility condition allowed all routers to correctly remove state of a tree, when the source stopped transmitting data packets. The use of this condition was crucial in this topology, due to the presence of a loop.

With all performed tests we have proved that the protocol worked as expected. We have verified that the protocol kept working, by maintaining trees consistently, even in the presence of router failures, reboots and network loops. The synchronization mechanism allowed a router that recovered from a failure to quickly exchange information with its neighbor routers, accelerating its recovery. All control messages were correctly built, being fully dissected by *Wireshark* with the implemented dissector.

References

[1] Netkit-NG. Online; accessed 14 November 2017.

[2] GNS3. Online; accessed 1 January 2018.

[3] TCPDUMP/LIBCAP public repository. Online; accessed 05 October 2018.

[4] Wireshark - Go Deep. Online; accessed 05 October 2018.

[5] Pedro Oliveira. Python implementation of IGMPv2, PIM-DM, and HPIM-DM. Internal Report, Instituto Superior Técnico, November 2018. `https://github.com/pedrofran12/hpim_dm/tree/master/docs/PythonImplementations.pdf`.

[6] William C. Fenner. Internet group management protocol, version 2. RFC 2236, RFC Editor, November 1997.

[7] Pedro Oliveira. HPIM-DM implementation. `https://github.com/pedrofran12/hpim_dm`, 2018. Online; accessed 16 September 2018.

[8] John Moy. Ospf version 2. STD 54, RFC Editor, April 1998.

[9] Paul Jakma. Quagga Routing Suite.

[10] Pedro Oliveira. PIM-DM implementation. `https://github.com/pedrofran12/pim_dm`, 2018. Online; accessed 16 September 2018.