Hard-state Protocol Independent Multicast – Source Specific Multicast (HPIM-SSM)

Margarida Marques Simões

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisor: Prof. Rui Jorge Morais Tomaz Valadas

Examination Committee

Chairperson: Prof. Ricardo Jorge Fernandes Chaves
Supervisor: Prof. Rui Jorge Morais Tomaz Valadas
Member of the Committee: Prof. Amaro Fernandes de Sousa

January 2021

Acknowledgments

I would like to thank my supervisor, Prof. Rui Valadas for his commitment and motivation in this work since day one. It was a very challenging work in a subject of big interest for me. During its development, he always showed me support and availability, and his guidance was crucial. I also would like to thank Pedro Oliveira for his dedication and contribution to this work. His insight in the Dissertation´s subject was very valuable. I also thank Instituto de Telecomunicações (IT) for all the research and projects developed in the area of Telecommunications.

I thank Instituto Superior Técnico for providing an enriching and challenging academic life that made me grow as a student and person. Special thanks to my family and friends for their unconditional support and encouragement during my academic life.

Abstract

In IP communications, multicast routing protocols provide an efficient way of distributing traffic from a node to a group of nodes in the network. This is achieved through a distribution system usually formed by one or more trees connecting the sources to the receivers through optimal paths. The main multicast protocols and most used currently are from the PIM family which includes the PIM-DM, PIM-SM, and PIM-SSM protocols. Due to their soft-state nature, PIM protocols suffer from slow convergence caused essentially by their slow reaction to events and changes in the network.

In this MSc Dissertation we developed a hard-state version of PIM-SSM, designated by HPIM-SSM, which overcomes several limitations of the current PIM-SSM protocol. We present the specification of HPIM-SSM. The protocol was implemented in Python and tested in a network emulated environment. The correctness of the specification was verified through model checking techniques using the Promela language and SPIN tool. We compared the convergence time of HPIM-SSM and PIM-SSM by creating and executing convergence time tests in both protocols. For this purpose we also implemented the PIM-SSM protocol in Python. Our results show that HPIM-SSM has much better convergence performance than PIM-SSM at the cost of some added stored information.

This MSc dissertation was supported by Instituto de Telecomunicações.

Keywords

IP Multicast; Multicast routing protocols; PIM; PIM-SSM.

Resumo

Em comunicações IP, os protocolos de encaminhamento multicast, permitem distribuir tráfego de forma eficaz de um nó para um grupo de nós na rede. Isto é alcançado através de um sistema de distribuição de tráfego geralmente formado por uma ou mais árvores que ligam as fontes aos receptores usando caminhos óptimos. Os principais protocolos multicast e mais usados hoje-em-dia pertencem à família PIM que incluí os protocolos PIM-DM, PIM-SM, e PIM-SSM. Devido à sua soft-nature, os protocolos PIM sofrem de convergência lenta, causada essencialmente pela sua reacção lenta a mudanças ou eventos na rede.

Nesta Dissertação de Mestrado desenvolvemos uma versão hard-state do protocolo PIM-SSM, designada por HPIM-SM, que supera muitas das limitações do PIM-SSM. Apresentamos a especificação do HPIM-SSM. O protocolo foi implementado em Python e testado num ambiente de emulação de redes. A correcção da especificação foi verificada através de técnicas de model checking, usando a linguagem Promela e a ferramenta SPIN. Comparámos os tempos de convergência do HPIM-SSM e PIM-SSM com a criação e execução de testes de convergência em ambos os protocolos. Dado isso, também implementámos o protocolo PIM-SSM em Python.

Os resultados obtidos mostram que o HPIM-SSM converge muito mais rápido que o PIM-SSM com o custo de se ter de guardar mais informação.

Esta tese foi apoiada pelo Instituto de Telecomunicações.

Palavras Chave

IP Multicast; Protocolos de encaminhamento multicast; PIM; PIM-SSM.

Contents

List of Figures

Acronyms

AL	Assert Loser
AW	Assert Winner
DR	Designated Router
HPIM-DM	Hard-state Protocol Independent Multicast - Dense Mode
HPIM-SSM	Hard-state Protocol Independent Multicast – Source Specific Multicast
IGMP	Internet Group Management Protocol
MLD	Multicast Listener Discovery
OSPF	Open Shortest Path First
PIM	Protocol Independent Multicast
PIM-DM	Protocol Independent Multicast - Dense Mode
PIM-SM	Protocol Independent Multicast - Sparse Mode
PIM-SSM	Protocol Independent Multicast - Source Specific Multicast
RP	Rendezvous Point
RPC	Root Path Cost

1 Introduction

Contents

Traditional IP communications support unicast transmissions where a host sends packets to a single host (one-to-one communications) and broadcast communications where a host sends packets to all hosts connected to a specific subnet. IP multicast communications allow a host to send IP packets to a subset of all hosts within the network. It provides a distribution system, usually formed by one or more trees for delivering multicast traffic from sources to groups of interested hosts. For applications that use group communication, e.g., videoconferencing, Internet TV distribution, stock quotes exchanges, or distance learning, the utilization of an IP multicast protocol is essential for efficient management of the resources of the network [1–3].

Currently, the main multicast routing protocols of the Internet are PIM-DM [4], PIM-SM [5], and PIM-SSM [5]. Between these three, the most popular among the telecommunication operators and more widely used are the PIM sparse mode protocols: PIM-SM and PIM-SSM [6]. However, all PIM protocols suffer from slow convergence due to the slow reaction to events such as link failures, or cost changes. These issues derive from the soft-state nature of PIM protocols that rely on the periodic transmission of control messages to keep the state of routers updated. In addition, there may be losses of the stored state in the routers, leading to network inconsistencies, unwanted removal of trees, and subsequently tree rebuildings. This can cause excessive network congestion and loss of multicast data. In previous work, a hard state version of PIM-DM designated by HPIM-DM [7] was developed. It keeps the main characteristics of PIM-DM but has a better convergence time, faster reconfiguration in the presence of network failures or unicast route changes, and message sequencing and reliability mechanisms.

In this work, we developed a hard-state version of PIM-SSM, designated by HPIM-SSM, that accomplishes the same that HPIM-DM did, faster convergence and faster reaction to network changes. HPIM-DM and HPIM-SSM guarantee that the trees have optimal paths from the receivers to the source. The tree states are stored in the routers and are only removed or changed when a control message or other specific event in the network happens. The routers do not rely on timers to keep their states updated or to transmit control messages. The transmission of control messages is triggered by specific events and is immediate. HPIM-SSM also has message sequencing and reliability mechanisms. Our work was based on the HPIM-DM work, since both protocols are hard-state and have the same type of mechanisms.

The correctness of the specification was verified through model checking techniques using the Promela language [8] and SPIN tool [9]. The protocol was implemented in Python based on its specification and its implementation was tested using NetKit-NG [10], a network emulated environment. PIM-SSM was also implemented in Python and this implementation was used to perform convergence time tests. These tests were also performed in HPIM-SSM, which allowed us to compare the convergence of both protocols regarding a group of network events.

1.1 Objectives

The goal of this MSc Dissertation is to specify, implement, and test a hard-state version of the PIM-SSM protocol, designated by HPIM-SSM, with faster convergence and faster reaction to network changes.

1.2 Organization of the Document

This document is divided into the following chapters:

1. **Introduction**: IP multicast importance and objectives of this work.
2. **State of Art**: Description of multicast routing principles and PIM multicast routing protocols, namely, PIM-DM, PIM-SM, PIM-SSM and HPIM-DM.
3. **HPIM-SSM specification**: Description of HPIM-SSM focusing on the main concepts of this new protocol;
4. **HPIM-SSM correctness tests**: Description of the correctness tests performed using model checking techniques.
5. **Protocol Implementations**: Description of the protocol implementations: PIM-SSM and HPIM-SSM.
6. **HPIM-SSM implementation tests**: Description of the implementation tests performed using a network emulated environment.
7. **Protocols convergence time and stored state**: Comparison of the convergence time and stored state of the HPIM-SSM and PIM-SSM protocols.
8. **Conclusions**: Main conclusions of the work developed.

The appendix includes:

1. **Appendix A**: HPIM-SSM Interest state machine
2. **Appendix B**: HPIM-SSM Assert state machine

2 State of Art

Contents

The first part of this chapter provides a brief and concise description of the multicast routing principles. The second part describes the most relevant multicast routing protocols for this work (PIM protocols).

2.1 Multicast Routing Principles

2.1.1 IP Multicast Addresses

IP multicast allows a host to send IP packets to a multicast group formed by a subset of all hosts in the network. This IP multicast group is identified by a multicast address that is the destination address in an IP multicast packet. The Internet Assigned Numbers Authority (IANA) assigned the IP addresses with the prefix 224.0.0.0/4 to be used for IPv4 multicast [11] and with the prefix FF00::/8 for IPv6 multicast [12].

2.1.2 IP multicast group membership protocols

Hosts manifest interest in receiving multicast traffic to the routers directly connected to the subnet they are attached to, by using the Internet Group Management Protocol (IGMP) for IPv4 networks or the Multicast Listener Discovery (MLD) protocol for IPv6 networks. There are three versions of the IGMP protocol and two versions of the MLD protocol. With IGMPv1 [13], IGMPv2 [14], and MLDv1 [15] a host is only capable of signaling from which groups it wants to receive the multicast traffic. IGMPv3 [16] and MLDv2 [17] add support for source filtering, i.e. allows a host to signal interest in receiving traffic from a certain source or all but specific sources.

2.1.3 Multicast Distribution Trees

Multicast distribution trees are used to control the path that multicast traffic takes through the network, so it reaches all hosts of a specific group. They save resources in the network by avoiding the creation of a separate path for each interested host. They must be reconfigured dynamically since the hosts can join or leave the group at any time. The two main types of trees used in multicast routing protocols are the source and shared trees [2, 18]. Source trees, also known as shortest-path trees (SPTs), have as root the source of the multicast traffic and span all hosts of a specific multicast group. They are referred to as (S, G) trees, where S and G are the addresses of the source and multicast group, respectively. Each pair (S, G) has a separate source tree. Shared trees, unlike source trees, do not have the root at the source but at some chosen point in the network, often called a Rendezvous Point (RP). Sources send their traffic to the RP, and the RP forwards this traffic downwards the shared tree to reach all hosts that joined a specific multicast group. They are referred to as (*, G) trees, where * means all sources, and G is the address of the multicast group. Multicast routers store the information about each tree in a multicast routing table. There is one multicast routing entry for each multicast source in the case of source trees and for each multicast group in the case of shared trees.

2.1.4 Multicast forwarding

Multicast routing tables indicate the interface from which packets must arrive (the root interface) and the interfaces used to forward packets (the non-root interfaces) for each IP multicast destination address.

Reverse Path Forwarding (RPF) The reverse path forwarding check is a fundamental mechanism in multicast routing that allows routers to correctly forward multicast traffic down the distribution tree by determining the root and non-root interfaces. When a multicast packet arrives at an interface, the router performs an RPF check to determine if it arrived through the root interface. The root interface is the one that provides the shortest path from the router to the source. If this is the case, the RPF check succeeds, and the packet is forwarded through the non-root interfaces. On the contrary, if the RPF check fails (because the packet did not arrive through the root interface), the packet is dropped. The RPF check helps to prevent loops and duplication of traffic in the distribution tree [18].

Multicast forwarding using dense and sparse mode Dense mode multicast routing protocols assume that most subnets in the network have at least on receiver interested in receiving multicast traffic [1]. Some dense mode routing protocols are DVMRP [19], PIM-DM, and M-OSPF [20]. Sparse mode routing protocols assume that no hosts want to receive multicast traffic until they specifically ask to receive it. Some sparse-mode routing protocols are PIM-SM, PIM-SSM and CBT [21].

2.2 Multicast Routing Protocols

In this section, we will describe the PIM protocols, namely, PIM-DM, PIM-SM, PIM-SSM and HPIM-DM.

2.2.1 PIM

The Protocol Independent Multicast (PIM) is not dependent on a specific unicast routing protocol, and it uses the information of the unicast routing table to perform the RPF check and create distribution trees. A router can only have one root interface for any entry in the multicast routing table. PIM routers do not send or receive routing updates, which reduces the overhead significantly in the network compared with other multicast routing protocols like DVMRP. PIM has two modes of operation: sparse mode (PIM-SM and PIM-SSM) and dense mode (PIM-DM).

Although having different strategies, the PIM protocols have the following mechanisms in common:

1. **Neighbour Discovery through the exchange of Hello messages** To establish and maintain PIM Neighbor adjacencies, every Hello Period (30 seconds), a router multicasts a Hello message on

all enabled interfaces. PIM Hello messages contain a Hold Time value (three times the Hello Period) that tells the neighbor when to expire the adjacency in case of failure. PIM adjacencies are refreshed upon the reception of a Hello message, and a neighbor is considered dead if no Hello message is received during the Hold Time Period.

2. **Use of PIM Assert on shared links** The PIM Assert process is used to stop the duplication of traffic on shared links by electing a single router responsible for forwarding the traffic, designated by Assert Winner (AW). The AW is the router with the best cost to the source. We designate this cost by Root Path Cost (RPC). The Assert is triggered when more than one interface is transmitting traffic to the link. The interfaces will then send an Assert message with their RPCs. All interfaces in the link hear the Assert messages. The router with the best RPC to the source becomes the AW and continues to forward the multicast traffic to the link, while the other routers prune their interfaces and become Assert Losers (ALs). The ALs store who the AW is and its RPC but do not store the RPC of any other routers. In case of a tie in the RPCs, the router with the highest IP address wins. Before the trigger of the Assert, interfaces are in the NoInfo (NI) state. Figure 2.1 illustrates the Assert mechanism in PIM.

3. **Use of Prune Override on shared links** The Prune Override process is used on shared links and prevents them from being pruned while having downstream routers still interested in receiving the multicast traffic. When a downstream router sends a Prune message to its upstream router, the Prune Override mechanism is triggered. All routers in the link hear the Prune message, and if they don't want the link to be pruned (because they are interested in the traffic), they must send a PIM Join message to override the Prune. When a router receives a Prune message, it starts a 3-second timer. If no override Join is received during this time, the timer expires, and the router prunes its interface, such that no traffic is forwarded to the link. The Join messages are also heard by everyone in the link. Figure 2.2 illustrates the override mechanism in PIM. In this case, R1 sends a Prune to R3, its upstream router, that starts a 3-second timer. R2 hears the Prune, and since it is still interested, it sends a Join to R3 to override the Prune. R3 stops the timer upon the reception of the Join and its interface is not pruned.

2.2.2 PIM-DM

Protocol Independent Multicast - Dense Mode (PIM-DM) [4] is meant for dense networks since it assumes that all subnets in the network have hosts interested in receiving multicast traffic. It uses source trees to distribute multicast traffic to the hosts, and these trees are built using a flood-and-prune behavior.

PIM-DM Source Distribution Trees

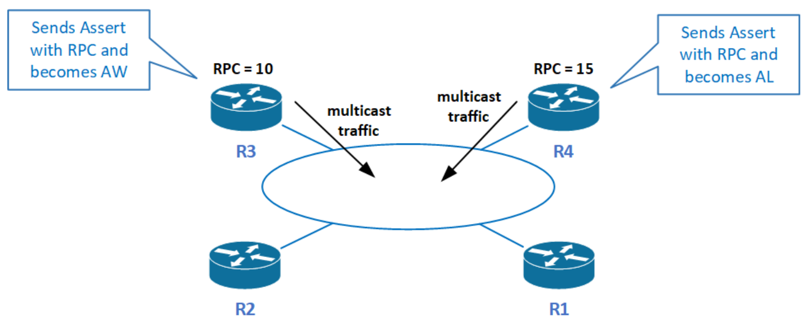

Sends Assert with RPC and becomes AW
RPC = 10
multicast traffic
multicast traffic
RPC = 15
Sends Assert with RPC and becomes AL
R3
R4
R2
R1

Figure 2.1: PIM Assert

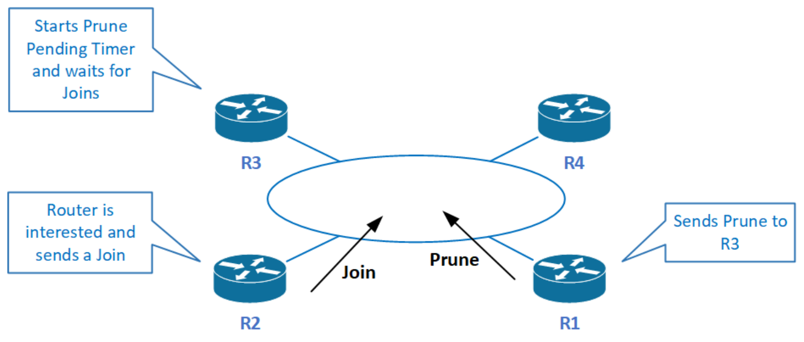

Starts Prune Pending Timer and waits for Joins
R3
R4
Router is interested and sends a Join
Join
Prune
Sends Prune to R3
R2
R1

Figure 2.2: PIM Override

The flooding behavior builds an initial tree spanning all routers, also referred to as a broadcast tree. The broadcast tree is a reverse shortest path tree supported on a unicast routing protocol. The construction of this tree is data-driven since it is triggered by a source S1 start transmitting multicast traffic for a group G. The traffic is then flooded to all routers in the network using the RPF technique (described in section 2.1.3). In the end, all routers will have in their multicast routing table an (S1, G) entry identifying the root and non-root interfaces for source S1.

PIM-DM Pruning

In the broadcast tree created by the flood behavior, some interfaces may be transmitting multicast traffic unnecessarily, consuming network and router resources. PIM Prune messages are used to prune the tree so that unnecessary traffic stops being forwarded. We can distinguish between two causes for multicast traffic being sent unnecessarily on a link:

1. **Existence of redundant paths on the tree**

   In the existence of redundant paths, routers receive traffic through their non-root interfaces. In point-to-point links, this causes the exchange of Prune messages which leads to pruning the link. In shared links, the Assert is triggered (described in section 2.2.1).

2. **Router not interested in receiving the multicast traffic**

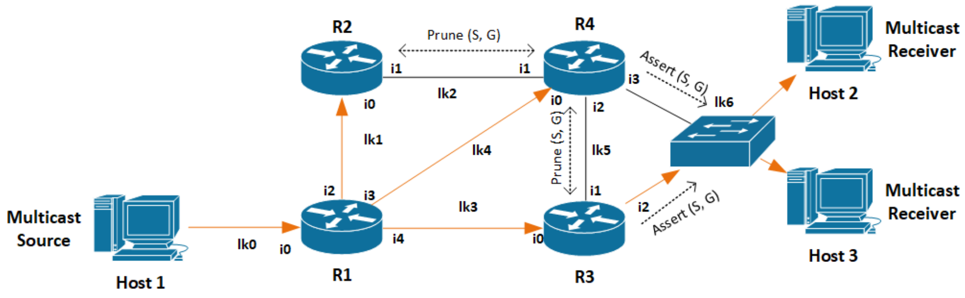


Figure 2.3: PIM-DM Source Tree

A router is not interested in receiving multicast traffic if it has no downstream interested neighbors or hosts. If a downstream router is not interested in receiving the traffic, it sends a Prune message to its upstream router. In point-to-point links, the neighbor prunes its interface upon the reception of the message. In multi-access links the Prune Override mechanism is triggered (described in section 2.2.1). This builds an initial (S, G) source tree that delivers the traffic from the source to the interested hosts.

We present now an example of the creation of a PIM-DM broadcast tree and then the pruning of redundant paths using the network topology of Figure 2.3. All interfaces i0 are root, and the remaining ones are non-root. Host 1 is the source, and hosts 3 and 4 are interested in receiving multicast traffic for group G. R2 and R3 have an RPC of 10, and R4 has an RPC of 15.

Suppose the source starts transmitting multicast traffic for group G. The traffic will be flooded to all routers in the network using the RPF technique (described in section 2.1.4). R1 floods the traffic through its non-root interfaces (i2, i3, and i4) reaching R2, R3, and R4. R2, R3, and R4 repeat the procedure done by R1. There is the duplication of traffic on links lk2, lk5, and lk6. On the shared link lk6, R4 and R3 receive traffic through their non-root interfaces, and the Assert is triggered (described in section 2.2.1). R3 is elected the AW since it has the best RPC to the source. On the point-to-point links lk2 and lk5, there is the exchange of Prune messages, and the links are pruned. The redundant paths have been removed from the broadcast tree, giving rise to a source tree where there is no duplication of traffic. In Figure 2.3, the source tree is represented by the orange arrows and the control messages transmitted by the dashed arrows.

PIM-DM Grafting

PIM-DM can graft back previously pruned branches of the distribution tree. A router sends a Graft message to an upstream neighbor when it had previously sent a Prune message to it and starts being interested in receiving the multicast traffic again. Upon the reception of the Graft message, the upstream neighbor changes its interface from the pruned to the forwarding state. Graft messages are protected through an ACK-protection mechanism [4].

PIM-DM State Refresh Mechanism

When a router receives a Prune message, it puts the respective interface in prune state and sets a Prune timer (3 minutes). If the timer expires, the interface goes back to the forwarding state, and multicast traffic starts being transmitted on it. PIM-DM has a State Refresh mechanism that prevents interfaces from being unpruned every 3 minutes [4]. First-hop routers (routers directly connected to the source) send periodic (S1, G) State Refresh messages (every minute) that are forwarded down the original broadcast tree, as long as the source keeps transmitting for group G. When a router receives an (S1, G) State Refresh message it resets the prune timers of its non-root interfaces associated with the (S1, G) state entry. This prevents pruned interfaces from timing out, and consequently, the reflood of traffic everywhere in the network.

2.2.3 PIM-SM and PIM-SSM

2.2.3.A Protocols Operation

Protocol Independent Multicast - Sparse Mode (PIM-SM) and Protocol Independent Multicast - Source Specific Multicast (PIM-SSM) are meant for sparse networks and work with a completely different strategy of PIM-DM. Trees are built from the bottom to the top where hosts are the bottom and the source the top. It is the interest of the hosts that triggers the creation of the trees. Routers explicitly request to receive multicast traffic using PIM Join messages that are sent towards the root of the tree. In PIM-SM, a shared tree is used to distribute multicast traffic to the receivers. The root of a shared tree is the RP. There is a shared tree for each active multicast group in the network. Sources must register with the RP to notify it that they are active through a source registration process. Then they can start transmitting multicast traffic to the RP that forwards it downwards the shared tree. PIM-SM enables a last-hop router to switch from the shared to the source tree for a specific source through the source tree switchover process. For each pair (S, G), a different source tree is built having as root the source.

Hosts manifest their interest in multicast traffic through the IGMP/MLD protocols. Since all routers in the link are notified, it is necessary to elect a single router responsible for sending this interest upwards. Otherwise, more than one router would do it and parallel paths to the source would be created. This router is called the Designated Router (DR). On every link is elected a DR using Hello messages. The DR is the router with the highest IP address at the link. The roles of the DR are: track local hosts interested in receiving multicast traffic, send Join/Prune messages towards the root node (RP or source), and, if no Assert is active, forward traffic to that shared link on behalf of any hosts.

When a host notifies the last-hop DR of its interest, the router creates a new entry in its multicast routing table and sends an (*, G) Join message to its upstream neighbor towards the RP as indicated

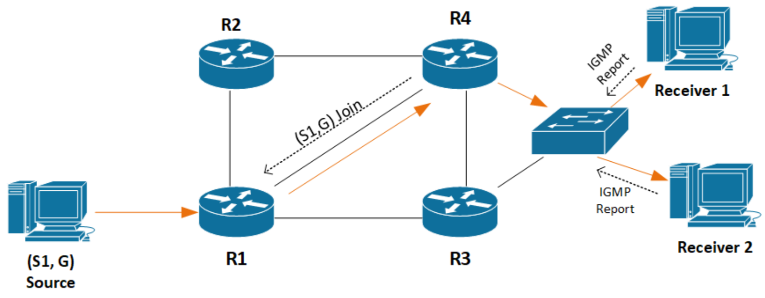


Figure 2.4: PIM Source Tree

in its unicast routing table. As this Join message travels upwards the tree, routers set the appropriate forwarding state so that the requested multicast traffic will be forwarded downwards the tree. If a router has no longer directly connected hosts or downstream routers interested in receiving multicast traffic for a group G, it sends an (*, G) Prune message towards the RP. As this Prune travels upwards the tree, routers update their forwarding state appropriately, pruning the tree. The process used to build and prune the PIM-SM source trees is identical to one of the shared trees, but instead of sending (*, G) Join and (*, G) Prune messages towards the RP, it sends (S1, G) Joins and (S1, G) Prunes towards the source S1.

Figure 2.4 shows an example of the construction of a source tree where both receivers manifested interest in (S1, G) multicast traffic via IGMPv3. Both R3 and R4 receive the interest of the hosts, but only R4 sends an (S1, G) Join message towards the source since it is the DR of the link. R4 creates an (S1, G) entry in its multicast routing table. R1 upon the reception of the Join, repeats the procedure done by R4, but no more Joins are transmitted since R1 is directly connected to the source. The solid orange arrows represent the final source tree. If the source is active, traffic is being forwarded following the path: S1-R1-R4-Receivers.

Derived from the soft state nature of PIM, the forwarding state of interfaces has a lifetime of 3 minutes, after which is deleted. This causes all already established trees to be torn down and having to be rebuilt. To maintain the forwarding state of interfaces, periodic (every minute) PIM Join messages are transmitted. A router maintains an interface in the forwarding state if it is not in the AL state and at least one of the following conditions is met: (i) a downstream router continues to send PIM Joins for the group, or (ii) a host connected to the local subnet continues to send IGMP Reports for the group. When one of the events occurs, the router refreshes the forwarding state of the interface and resets the 3-minute timer.

There is a subset of the PIM-SM protocol that does not include the shared tree, the RP, and the registration of the sources with the RP, designated by PIM-SSM [5]. The PIM-SSM protocol inherits the Assert and Prune Override mechanisms and the control messages of PIM-SM (Hello, Assert, Join, and

Prune). While in PIM-SM hosts manifest interest for a multicast group, in PIM-SSM they also need to indicate a source by using the IGMPv3 or MLDv2 protocol since only (S, G) trees are used and these are built from the bottom up. The drawback of PIM-SSM is that trees can be built without the sources being active, consuming resources in the routers to maintain them. In PIM-SM/PIM-SSM, the non-root interface (S, G) state machine has the following states: NoInfo (NI): the interface has no interest state and no timers are running; Join (J): the interface has interest state and will forward (S, G) packets except if it is AL; Prune-Pending (PP): the interface received a Prune and is waiting to see if other downstream routers are still interested (prune override mechanism). During this state, the interface is forwarding like it was in the Join state. In PIM-SSM, the root (S, G) state machine has the following states: NotJoined (NJ): no non-root interface has (S, G) forwarding state; Joined (J): at least one non-root interface has (S, G) forwarding state.

2.2.3.B Relation of the Designated Router with the Assert

To better understand the creation and maintenance of source and shared trees we study more in depth the relation of the DR with the Assert.

On every link is elected a DR and this election is based on IP addresses. It may happen that the elected DR is not the router with the shortest path to the root. This is an issue since trees can be created without having the optimal path from the hosts to the root. This situation can only be corrected with the trigger of the Assert and election of the AW, which rebuilds the tree with the optimal paths. In PIM-DM there is always the election of an AW. Trees are created when a source start transmitting traffic and this traffic is initially flooded to all routers in the network, triggering the Assert on the shared links. In PIM-SM/PIM-SSM it may happen that the Assert is not triggered because of the existence of the DRs and the trees being created from the bottom up. If the Assert is not triggered, the tree will not be reconfigured and so, traffic will be forwarded from the source to the hosts, not through an optimal path.

Triggering the Assert in PIM-SM and PIM-SSM

In PIM-SSM/PIM-SM, the Assert is triggered when more than one router is transmitting traffic to the link and this only happens under certain circumstances. Two network scenarios can trigger the assert and these are:

Scenario 1: The first scenario is when we have a shared link with at least two downstream interested routers and their unicast routing tables have different information regarding the next-hop router for a specific source. This inconsistency can be temporary and caused by delays in the update of the unicast routing tables. It also can be caused by the routers having different static routes to a specific source. When this inconsistency happens, the downstream routers may send Join to different upstream routers, and this interest is propagated upstream by more than one router. If a source is active, more than one router will be transmitting multicast traffic in the link and the assert is triggered.

Scenario 2: Another possible scenario is when we have a shared link with interested downstream routers and hosts, and more than one upstream router, with the DR not being the router with the best RPC to the source. The DR is the router responsible for propagating the interest of the hosts upstream but since it is not the router with the best RPC, the downstream routers send Join messages to the router with the best RPC (their next-hop router in the unicast routing table). In this case, we have more than one router propagating the interest upstream (the DR and the one with the best RPC), and so transmitting multicast traffic in the link. The assert is triggered and an AW is elected.

The router with the best RPC is elected the AW and the others AL. The AW starts sending Join upwards and rebuilds the tree. In this scenario, when the DR goes to the AL state, it stops forwarding traffic and sends a Prune upwards. The DR can only forward and send Joins (to maintain the tree) when it is in the NI or AW state. There is an assert timer (AT) with a periodicity of 180 seconds that is used to time out the Assert state on the ALs and to resend asserts on the AW. When the (AT) timer expires, the DR which is in the AL state will go to the NoInfo state and the AW will send an Assert. If the timer of the DR expires first, the router will have forwarding state again and so, it can transmit traffic until receiving the Assert from the AW. In the worst-case scenario, this behavior repeats every 180 seconds, creating high instability in the network by the periodic reconstruction of the tree and duplication of traffic.

GNS3 Assert Study

To better understand the connection between the DR and the Assert in PIM-SM/PIM-SSM, and how this relation can create instability in the network, we made a study using the GNS3, a network simulator, with Cisco Routers (Cisco c3725 routers) running PIM-SSM. First, we implemented the two scenarios described above to verify the Assert was triggered. The network topology used for the first scenario is shown in Figure 2.5 and for the second scenario in Figure 2.6. Then, with the scenario 2 implementation, we study the creation, maintenance and periodic reconstruction of trees with focus on the DRs, the Join/Prune mechanisms, and its relation with the Assert. We observed the changes of the states in the routers, the messages exchanged, and changes in trees, by placing Wireshark captures in the links. Both implementations, including the configurations of the routers and hosts and the Wireshark captures, can be accessed in our GitHub repository [22].

Now, we look with more detail into scenario 2 implementation. Our initial scenario is: all routers are running the PIM-SSM multicast protocol and the Open Shortest Path First (OSPF) as the unicast routing protocol. Hosts manifest their interest through the IGMPv3. R2 is the upstream router in the link with the best RPC to the source and R3 is the DR. Host 1 and 2 are interested in receiving multicast traffic for the tree (S, G). First, we verified that a tree was built including routers R4, R3, R2, and R1.

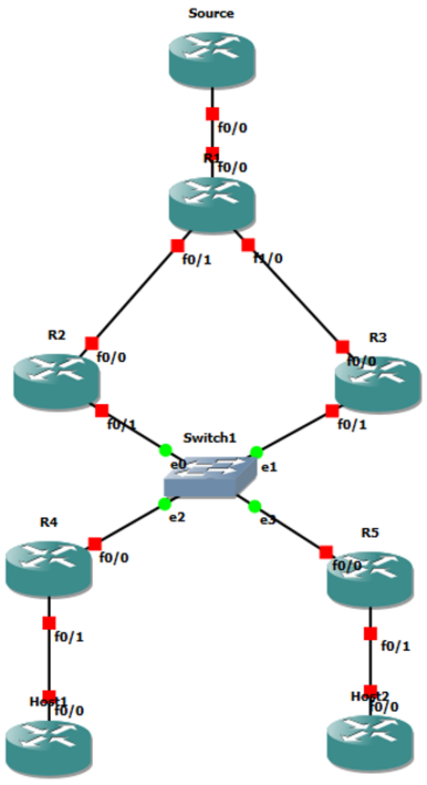


Figure 2.5: Network scenario 1

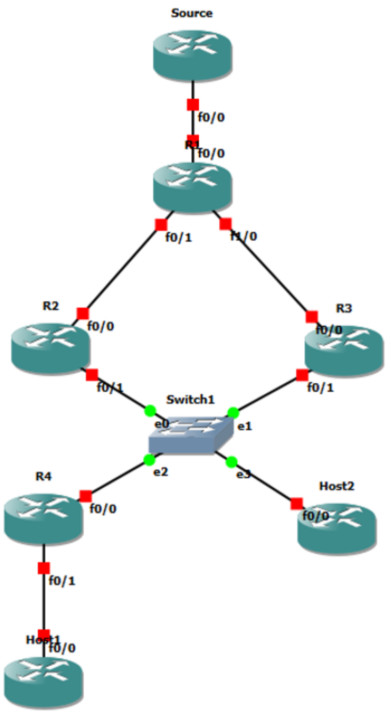
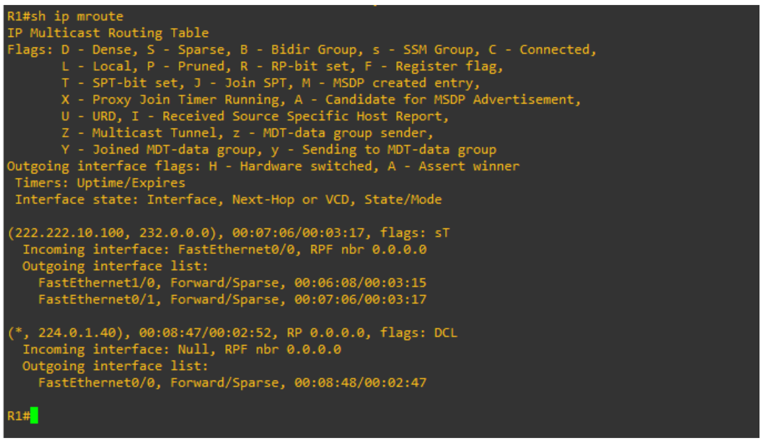


Figure 2.6: Network scenario 2



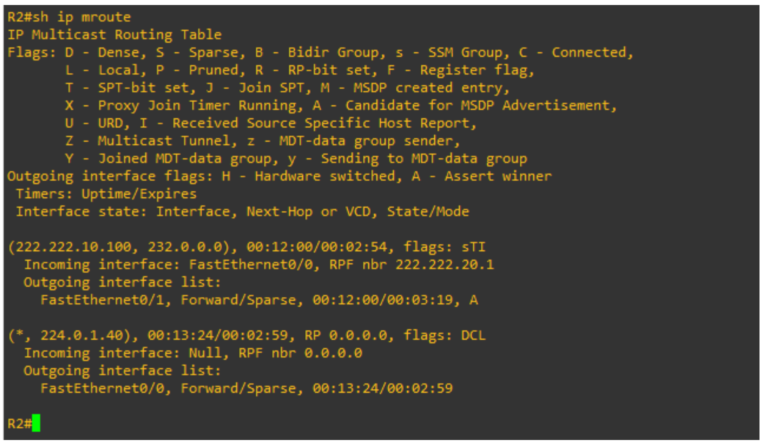
```
R1#sh ip mroute
IP Multicast Routing Table
Flags: D - Dense, S - Sparse, B - Bidir Group, s - SSM Group, C - Connected,
       L - Local, P - Pruned, R - RP-bit set, F - Register flag,
       T - SPT-bit set, J - Join SPT, M - MSDP created entry,
       X - Proxy Join Timer Running, A - Candidate for MSDP Advertisement,
       U - URD, I - Received Source Specific Host Report,
       Z - Multicast Tunnel, z - MDT-data group sender,
       Y - Joined MDT-data group, y - Sending to MDT-data group
Outgoing interface flags: H - Hardware switched, A - Assert winner
 Timers: Uptime/Expires
 Interface state: Interface, Next-Hop or VCD, State/Mode

(222.222.10.100, 232.0.0.0), 00:07:06/00:03:17, flags: sT
  Incoming interface: FastEthernet0/0, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet1/0, Forward/Sparse, 00:06:08/00:03:15
    FastEthernet0/1, Forward/Sparse, 00:07:06/00:03:17

(*, 224.0.1.40), 00:08:47/00:02:52, RP 0.0.0.0, flags: DCL
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet0/0, Forward/Sparse, 00:08:48/00:02:47

R1#
```

Figure 2.7: R1 multicast routing table



```
R2#sh ip mroute
IP Multicast Routing Table
Flags: D - Dense, S - Sparse, B - Bidir Group, s - SSM Group, C - Connected,
       L - Local, P - Pruned, R - RP-bit set, F - Register flag,
       T - SPT-bit set, J - Join SPT, M - MSDP created entry,
       X - Proxy Join Timer Running, A - Candidate for MSDP Advertisement,
       U - URD, I - Received Source Specific Host Report,
       Z - Multicast Tunnel, z - MDT-data group sender,
       Y - Joined MDT-data group, y - Sending to MDT-data group
Outgoing interface flags: H - Hardware switched, A - Assert winner
 Timers: Uptime/Expires
 Interface state: Interface, Next-Hop or VCD, State/Mode

(222.222.10.100, 232.0.0.0), 00:12:00/00:02:54, flags: sTI
  Incoming interface: FastEthernet0/0, RPF nbr 222.222.20.1
  Outgoing interface list:
    FastEthernet0/1, Forward/Sparse, 00:12:00/00:03:19, A

(*, 224.0.1.40), 00:13:24/00:02:59, RP 0.0.0.0, flags: DCL
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet0/0, Forward/Sparse, 00:13:24/00:02:59

R2#
```

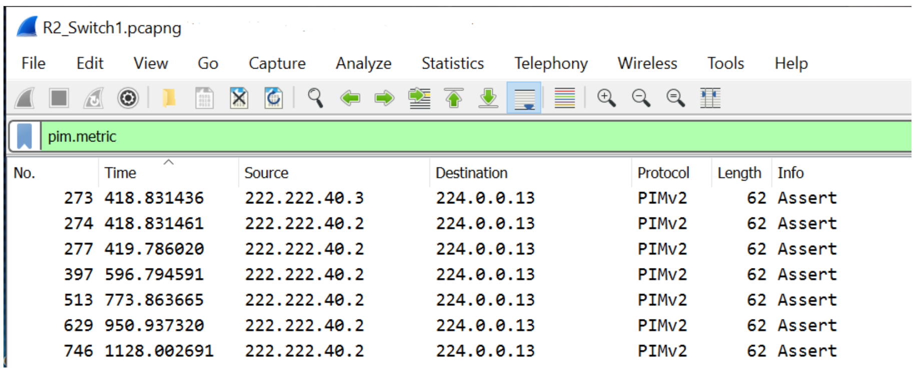
Figure 2.8: R2 multicast routing table

Remember that the DR, R3, transmits the interest of host 2 upstream, and the router with the best RPC, R2, transmits the interest of R4. The Figure 2.7 shows the multicast routing table of R1, with the two non-root interfaces forwarding.

When the source started transmitting for group G, R2 and R3 transmitted multicast data to the shared link and the assert was triggered as predicted. Since R2 is the upstream router with the best RPC, it is elected AW and the one responsible for transmitting the traffic to the link. R3 stops transmitting. The Figure 2.8 shows the multicast routing table of R2, with its non-root interface forwarding and in the AW state.

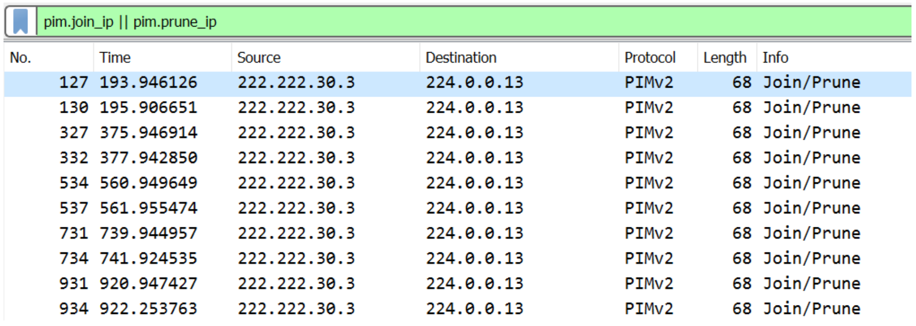
There are some issues we can already observe. First, the DR is not the router with the best cost to the source and creates a tree without an optimal path to the source on behalf of Host 2. Then, R4 also creates a tree because of the interest of R4, and parallel paths are created. This situation is only corrected when the source starts transmitting and an AW is elected.

After the AW election, it was expected for the routers to stay in the same states for 180 seconds.



```
No.    Time         Source         Destination    Protocol  Length  Info
   273 418.831436   222.222.40.3   224.0.0.13     PIMv2         62  Assert
   274 418.831461   222.222.40.2   224.0.0.13     PIMv2         62  Assert
   277 419.786020   222.222.40.2   224.0.0.13     PIMv2         62  Assert
   397 596.794591   222.222.40.2   224.0.0.13     PIMv2         62  Assert
   513 773.863665   222.222.40.2   224.0.0.13     PIMv2         62  Assert
   629 950.937320   222.222.40.2   224.0.0.13     PIMv2         62  Assert
   746 1128.002691  222.222.40.2   224.0.0.13     PIMv2         62  Assert
```

Figure 2.9: Assert messages in shared link



```
No.    Time         Source         Destination    Protocol  Length  Info
   127 193.946126   222.222.30.3   224.0.0.13     PIMv2         68  Join/Prune
   130 195.906651   222.222.30.3   224.0.0.13     PIMv2         68  Join/Prune
   327 375.946914   222.222.30.3   224.0.0.13     PIMv2         68  Join/Prune
   332 377.942850   222.222.30.3   224.0.0.13     PIMv2         68  Join/Prune
   534 560.949649   222.222.30.3   224.0.0.13     PIMv2         68  Join/Prune
   537 561.955474   222.222.30.3   224.0.0.13     PIMv2         68  Join/Prune
   731 739.944957   222.222.30.3   224.0.0.13     PIMv2         68  Join/Prune
   734 741.924535   222.222.30.3   224.0.0.13     PIMv2         68  Join/Prune
   931 920.947427   222.222.30.3   224.0.0.13     PIMv2         68  Join/Prune
   934 922.253763   222.222.30.3   224.0.0.13     PIMv2         68  Join/Prune
```

Figure 2.10: Periodic reconstruction of tree

When the Assert Timer expires there is a new AW election and exchange of Assert messages. If the AT expires first in the DR, there is also a reconstruction of the tree since the non-root interface of the DR can have forwarding state again during a short period. When the AT expires in the DR, the router goes to the NoInfo state and since it has hosts interested, it has forwarding state. Given that, it sends a Join upwards the tree and starts transmitting traffic. When the DR receives the Assert message from the AW (sent when the AT timer expires) it goes to the AL state, sends a Prune upwards, and stops forwarding.

During some minutes we observed the changes in the network and verified every 180 seconds this behavior occurred. Figure 2.9 shows the transmission of the Assert messages in the shared link. Figure 2.10 shows the Joins and Prunes sent by the DR (R3) every time its Assert Timer expires. Both figures are from the Wireshark captures placed in the link R2-switch and R1-R3.

The first three messages (No. 273, 274, and 277) of Figure 2.9 correspond to the first AW election, where initially R2 and R3 consider themselves the AW, and then R3 stays the effective AW. Then, every 180 seconds there is the reconstruction of the tree. The DR goes to the NI state and sends a Join upwards. When receives the Assert from the AW (1 or 2 seconds later), it goes back to the AL state and sends a Prune upwards. The transmission of a Join message and almost immediately a Prune message can be seen in Figure 2.10.

Messages with No. 127 and 130 correspond to the first reconstruction of the tree, where the message with No. 127 is a Join and No. 130 a Prune. After 180 seconds approximately, we have another reconstruction of the tree, messages with No. 327 (Join) and No. 332 (Prune). In Figure 2.10 we see five periodic reconstructions of the tree. This periodic behavior creates high instability in the network and can be identified as an issue of the PIM-SM/PIM-SSM protocols.

2.2.4 PIM-SM and PIM-SSM issues

2.2.4.A Lack of protection in control messages

One of the problems in PIM is the absence of protection in some control messages. The routers do not know if the messages they sent are delivered to the intended receivers, given that messages can be lost in the network. We present now an example showing the possible consequences of the lack of protection of Join messages in the Prune Override Mechanism. PIM routers expect to receive overriding Joins from downstream neighbors that wish to continue receiving traffic in response to a Prune sent by other neighbors in the link. If overriding Join messages are lost or delayed in the network, and the AW does not receive these messages before the Prune override timer times out, it goes to the NoInfo state and the traffic ceases to be transmitted on the link while one or more routers are still interested in receiving it.

2.2.4.B Lack of message ordering guarantee

Besides the lack of protection in control messages, there is also no guarantee that the transmission order is preserved. If a downstream router sends a Prune to the AW and then sends a Join, but the Prune is the last one to arrive, the AW is pruned, and the router stops receiving traffic despite still being interested. If the opposite occurs, the router sends first a Join and then a Prune, but the messages are received in the inverse order, the AW will keep forwarding traffic with no downstream interested routers.

2.2.4.C Slow Tree Reconfiguration

When an event that causes a tree reconfiguration occurs, e.g. the RPC changing in a router, the reconfiguration of the tree may only happen through the periodic reconstruction of the tree (which can take 3 minutes) or through the reception of periodic messages (around 1 minute). During this time, the tree may not have the best configuration making multicast traffic not being forwarded through the optimal paths between routers and the source.

2.2.4.D Designated Routers on shared links

The DR in a shared link is the router responsible for sending Prune and Joins messages on behalf of hosts. The DR is the router with the highest IP address at the link and not the one with the shortest path to the root node. If a host in a shared link joins a multicast group, a path will be installed between the DR of the link and the root node, to deliver the multicast traffic to the interested host. If the DR is not the router in the link with the shortest path to the source, the path installed will not be the optimal one.

2.2.5 HPIM-DM

Hard-state Protocol Independent Multicast - Dense Mode (HPIM-DM) is a hard-state version of PIM-DM that was developed on a previous MSc Dissertation [7]. Like PIM-DM, it is meant for dense networks, and it is IP routing protocol-independent. One main difference from PIM-DM is that routers maintain knowledge of the existing multicast trees at all times by keeping information about all upstream routers from which multicast traffic can be received. Another difference is the implementation of a synchronization process that allows a router joining the network to immediately learn which trees are active and thus start receiving multicast traffic if interested.

2.2.5.A Hello Protocol

In HPIM-DM, neighbor detection is performed by any control message and not just Hello messages, which speeds up the protocol convergence. The establishment of a relationship between two neighbors requires that they synchronize with each other and exchange tree information. The Hello protocol is the mechanism used to maintain neighborhood relationships between routers of an HPIM-DM network and is similar to the one used in PIM (described in section 2.2.1).

2.2.5.B Broadcast Tree Maintenance

Similarly to PIM-DM, HPIM-DM builds and maintains a broadcast tree by relying on a unicast routing protocol that determines at a router the root and the non-root interfaces for a given tree. The broadcast tree is a reverse shortest-path tree that allows the delivery of multicast traffic from a source to all network routers. There are two types of routers in a broadcast tree: originators and non-originators. The originators are the routers directly attached to the source, and all the rest are the non-originators.

2.2.5.C Broadcast Tree States

An HPIM-DM router must store two types of states regarding the broadcast tree: tree state and upstream state of each neighbor. A neighbor is UPSTREAM if multicast data can be received from it and

NOT UPSTREAM otherwise. To set the upstream state of neighbors, routers multicast upstream messages through their non-root interfaces. An IamUpstream message signals downstream routers that the sending router can serve as an upstream neighbor, and an IamNoLongerUpstream message signals the opposite. A router will only transmit an IamUpstream message downwards the tree if it has a parent on the tree. The parent of a router is the UPSTREAM neighbor that provides the lowest RPC to the source and has a lower RPC than the router itself (to avoid routing loops). A router transmits an IamNoLongerUpstream message when it loses its parent on the tree. Regarding the broadcast tree, a router can be in the ACTIVE, INACTIVE, or UNSURE state. It is in the ACTIVE state if it considers the source active and in the INACTIVE state if it considers the source inactive . The router is in the UNSURE state if it is not sure whether the source is active or not. A router keeps information on the broadcast tree as long as the source is active, without the periodic transmission of control messages or multicast traffic (hard-state nature).

2.2.5.D Installing and removing a Tree

When the source becomes active, the originators start receiving multicast data and send IamUpstream messages through their non-root interfaces. The messages are forwarded down the tree. In this way, a connection between routers and originators is established through a chain of parent routers. When the source becomes inactive, the same process takes place, but now, originators send an IamNoLongerUpstream message through their non-root interfaces. These messages are forwarded down the tree, removing the tree state stored in the routers.

2.2.5.E Assert Winner election

It is necessary to elect an AW in every link of the network to avoid the duplication of traffic. The process used to elect the AW is similar to the one used in the PIM Assert (described in section 2.2.1). In HPIM-DM, interfaces know the RPC of their upstream neighbors because this information is contained in the IamUpstream messages. Both root and non-root interfaces must be aware of who the AW is, although only non-root interfaces have Assert state (AW or AL).

We present now an example of the creation and removal of the broadcast tree and AW election using the network topology of Figure 2.11 where host 1 is the source, router A is an originator, all interfaces i0 are root and the remaining are non-root. Suppose router B and C have an RPC of 10 and router D has an RPC of 15.

If the source becomes active, the originator changes to the ACTIVE state and sends an IamUpstream message through its non-root interfaces (i2, i3 and i4). Routers B, C, and D save router A as an UPSTREAM neighbor and send an IamUpstream message though their non-root interfaces since they have now a parent on the tree (router A). This IamUpstream messages will not be forwarded again since

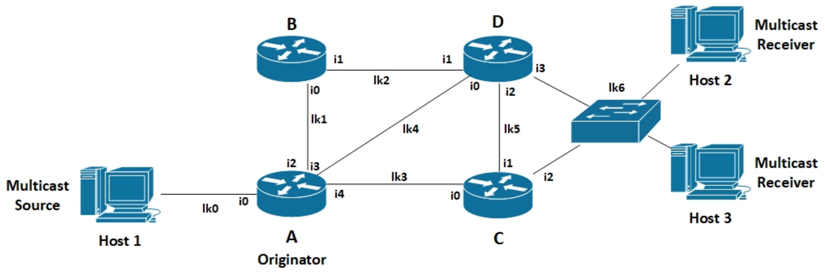


Figure 2.11: Broadcast Tree Maintenance

they will be received through non-root interfaces but the neighbors that sent them will be set as being UPSTREAM. Namely, router D sets B and C as UPSTREAM. Router B and C set D as UPSTREAM. Router A will become the AW in links lk1, lk3 and lk4 since it was the only router sending IamUpstream messages in these links. Router B is the AW of link lk2 since it has a better RPC than router D. In links lk5 and lk6, router C is AW for the same reason. If the source becomes inactive, the originator changes to the INACTIVE state and transmits an IamNoLongerUpstream message through its non-root interfaces (i2, i3 and i4). Router B, C, and D set router A as NOT UPSTREAM and send an IamNoLongerUpstream message since they lost their parent on the tree. All routers will stop having UPSTREAM neighbors and change to the INACTIVE state (the tree state is removed).

2.2.5.F Multicast Tree maintenance

The broadcast tree must be pruned from routers not interested in receiving multicast traffic, giving rise to a multicast tree spanning only the interested receivers.

In respect to the interest maintenance, some state information must be defined. Regarding the downstream interest, a non-root interface is in the DOWNSTREAM INTERESTED state if it has downstream interested neighbors or hosts and in the NOT DOWNSTREAM INTERESTED state otherwise. A non-root interface is in FORWARDING state if it is both the AW and DOWNSTREAM INTERESTED and is PRUNED otherwise. Interfaces in the FORWARDING state forward multicast data and in the PRUNED state do not. Regarding the interest in receiving multicast traffic from an (S, G) tree, a router can be in the INTERESTED or NOT INTERESTED state. A router is INTERESTED if it has at least one non-root interface in the FORWARDING state and is NOT INTERESTED otherwise. Multicast receivers signal their interest to the routers directly connected to them through the IGMP/MLD protocols. The interest of a router in receiving multicast traffic is signaled to its neighbors by the transmission of Interest and NoInterest messages. Interest messages are sent through root interfaces and signal upstream neighbors of the interest of a router in receiving multicast traffic. NoInterest messages are sent through root or non-root interfaces and signal the neighbors that the router is not interested in receiving multicast traffic. Within a link, the interest messages are unicasted to the AW, since only the AW needs to have the interest information updated to determine if it becomes FORWARDING or PRUNED.

2.2.5.G Installing and Maintaining a Multicast Tree

When a source becomes active and starts transmitting multicast data, the IamUpstream messages start flowing through the network. On every link, when the first IamUpstream message is transmitted, all other interfaces connected to the link react by sending their interest information, which leads to the formation of a multicast tree. The interest information must also be sent when the interest of a router changes so that the multicast tree stays updated. When multicast traffic is forwarded down the multicast tree, is received by root interfaces and re-transmitted by non-root interfaces in the FORWARDING state. If the tree is correctly formed, there is no duplication of traffic on the links.

2.2.5.H Message sequencing

HPIM-DM requires that the control messages are processed at the routers in the order they were transmitted. This is achieved by using a message sequencing mechanism. Message sequencing is performed locally between neighbors, and messages are numbered using sequence numbers (SN) according to the transmission order. The sequence numbers start at one, are incremented by one whenever a new message needs to be transmitted, and end at a predefined value depending on the number of bits used. An SN restart event occurs when the SN reaches its maximum value or when a router reboots. Messages also carry the BootTime value that determines the last SN restart event. The BootTime can be (i) the router's clock or (ii) a number that is persistently stored in the router and incremented, when a SN restart event occurs. When the value of BootTime is updated, the next SN being used will be the initial SN value. When comparing two messages, the one with higher BootTime is always fresher. If both messages carry the same BootTime the one with the highest SN is fresher. The SN received from neighbors must be stored, so later routers can compare the stored SNs with the SN contained in the newly received messages and evaluate their freshness. To reduce the amount of SNs stored at each router, a new SN was created: CheckpointSN. The CheckpointSN is transmitted periodically to the neighbors and indicates the highest SN that has been acknowledged so far. When a router receives the CheckpointSN from a neighbor, stores it, and removes all stored SNs lower than it.

2.2.5.I Synchronization process

Synchronization is a process between two neighboring routers, where there is the exchange of upstream information, to determine for which active trees the neighbor considers itself being UPSTREAM. It is triggered when a new router joins the network, a known router reboots or bidirectional communication is interrupted.

Consistency of Synchronization

During the synchronization process, each router sends to the neighbor the list of trees for which it considers being UPSTREAM regarding that neighbor. This information is exchanged in Sync messages, and its transmission must be protected. One or more Sync messages may be needed to transmit all information. The information exchanged is only valid when the synchronization process ends.

The synchronization starts by the router taking a snapshot of all trees for which it considers being UPSTREAM regarding the neighbor. The snapshot is then transmitted to the neighbor even if the upstream state of the router changes. A synchronization period is identified with two sequence numbers: SnapshotSN (created by incrementing the SN of the transmitting interface by one) and BootTime.

Sync messages include the BootTime and SnapshotSN of the sending router and the neighbor to which it is synchronizing. This ensures that both neighbors know the BootTime and SnapshotSN of each other and that the Sync messages exchanged are for sure from the same synchronization period. Moreover, the SnapshotSN allows ignoring all upstream and interest messages of all trees received with an SN equal or lower than the SnapshotSN. If the router during synchronization changes its upstream state relatively to any tree, it may trigger the transmission of control messages to the neighbor. These messages will be accepted by the neighbor, given their SN being higher than the SnapshotSN.

Synchronization protocol

The synchronization protocol is similar to the one used in OSPF [23]. One router is elected MASTER, having the responsibility of controlling the communication process. The transmission of Sync messages is protected by a Stop-and-Wait protocol controlled by the Master. Sync messages are numbered using the sequence number SyncSN to identify different snapshot fragments. The Master stops the synchronization when the reception of all fragments has been acknowledged.

2.2.5.J Reliable message transmission

There are three types of messages for which we must ensure reliability: hello, upstream/interest, and Sync messages. Hello messages are reliable due to their periodicity. Sync messages are protected using the Stop and Wait protocol, which is based on the synchronization mechanism used by OSPF. An ACK-protection mechanism reliably protects upstream/interest messages, such as interest and upstream messages. All control messages have an SN. When a router sends a message, sets a timer, and waits to receive an ACK message with the same SN. If the time expires and no ACK message is received, this process repeats until the message is successfully acknowledged.

3

HPIM-SSM Specification

Contents

Hard-state Protocol Independent Multicast – Source Specific Multicast (HPIM-SSM) protocol is a hard-state version of PIM-SSM meant for sparse networks, that aims to react faster to network changes, and have better resilience to replay attacks and reliability in the transmission of control messages. In section 2.2.4 we presented some issues of the PIM-SM/PIM-SSM protocols that we propose to solve or at least minimize. In the specification, we will discuss the type of control messages, and the states needed to be stored in the routers. Besides the tree maintenance, we will cover the synchronization process and the reliability and sequencing of the messages.

3.1 Protocol Overview

Before entering into the details of the protocol, we present an overview of its main concepts and ideas. As PIM-SSM, HPIM-SSM is an IP multicast routing protocol that relies on a unicast routing protocol to perform the RPF checks. The creation of trees is also triggered by hosts manifesting interest in a specific source and group using the IGMPv3/MLDv2 protocols. With a hard-state version of PIM-SSM, we ensure a faster convergence of the protocol and eliminate the need of periodic transmission of Join messages, the periodic reconstruction of trees, and the delays caused by timers. To built a hard-state version we need to ensure that control messages are received and processed in the correct order in each router. Like PIM-SSM, HPIM-SSM uses the Join/Prune messages to create the source trees and the Assert to elect a single forwarder in each link, which is also responsible for sending the interest upwards. In HPIM-SSM, the Assert is triggered by the arrival of interest at a link and this interest can be of routers or hosts. With this, we avoid the need for the DRs and all the problems that PIM-SM/PIM-SSM suffer from the relation of the DR with the Assert (explained in section 2.2.3.A). An interface can only forward and send interest upwards if it is in the AW state, and in every link, an AW must be elected. In the presence of interest, the non-root interfaces can be AWs and the root interfaces can become non-root, and therefore, potential AWs. Given that, all interfaces need to store the interest and the RPC of their neighbors. Due to the hard-state nature of the protocol, when a router joins the network it is necessary a synchronization process with its neighbors so it can learn information about the existent trees in the network.

Figure 3.1 illustrates the main concepts of the protocol described above. It shows a network scenario where initially R1, R3, and R4 are running HPIM-SSM and both Receivers are interested in (S1, G) multicast traffic. R4 has a better RPC than R3. When the receivers manifest their interest, R3, and R4 save it, the Assert is triggered, and R4 is elected AW since it has better RPC. R4 sends the interest to its next hop router´in the unicast routing table for the source S1. When R1 receives the interest, saves it, and becomes AW. It does not send the interest upwards since it is directly connected to the source. The

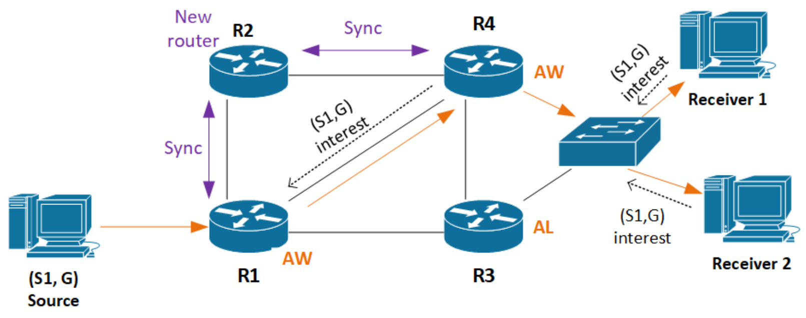


Figure 3.1: HPIM-SSM Overview

orange arrows represent the tree created and the path used to forward the (S1, G) traffic from the source to the receivers. After the tree being created, R2 starts the HPIM-SSM protocol and synchronizes with its neighbors (R4 and R1) to learn information about the active trees in the network. Each purple arrow represents the synchronization process between two neighbors.

In section 3.2 we explain the Hello protocol, in section 3.3 the concepts of creating and maintaining trees, and in section 3.4 and 3.5 the interest and assert state machines respectively. Section 3.6 covers the removal of tree state, section 3.7 the installation and removal of a tree in the network, and section 3.8 the synchronization process and the information that needs to be exchanged initially. In section 3.9 is covered the message sequencing and reliability mechanisms and in section 3.11 the message's format. Finally, in section 3.11 is covered the existence of loops in the HPIM-SSM protocol and in section 3.12 we present some examples of the HPIM-SSM.

3.2 Hello Protocol

The Hello protocol in HPIM-SSM is the same as in HPIM-DM and is described in section 2.2.5.A.

3.3 Tree Formation and Maintenance Concepts

Two types of messages are used in the tree formation and maintenance procedure: (i) Assert messages used in all links to elect the AW and (ii) Join/Prune messages to inform neighbors about the interest of a router in joining or leaving an (S, G) source tree. Assert, Join, and Prune messages are multicasted between neighbors (link-local scope). When a router receives a notification of interest on a non-root interface and the interface becomes AW, it propagates the interest towards the source of the tree. A router will indefinitely consider a downstream neighbor being interested until it advertises otherwise. A tree is removed when there are no interested receivers left for the corresponding source by the transmission of

Prune messages towards the source.

3.4 HPIM-SSM Interest

The interest/no interest of the hosts triggers the creation/removal of the trees. Interfaces know if there are any hosts interested with the IGMPv3/MLDv2 protocol. The creation/removal of trees is done by sending Join/Prune messages upwards until reaching a router directly connected to the source. The Join message indicates to the receiving router that the sending router is interested in receiving multicast traffic from a specific (S, G) tree, and the Prune message indicates the opposite. When an interface receives a Join (S, G) message from a neighbor, it stores the interest of the neighbor in receiving multicast traffic from the (S, G) tree. This information is only removed upon the reception of a Prune (S, G) message from the same neighbor. Routers send Join/Prune messages when there is a change in their interest state. Both Join and Prune messages are multicasted through root interfaces and include the IP address of the source (S) and the multicast group (G).

Root interfaces receive multicast traffic according to the interest state of the router, and non-root interfaces transmit the multicast traffic according to their forwarding state. The interest state of a router depends on the forwarding state of its non-root interfaces. The forwarding state of a non-root interface depends on its Assert state. The Assert states will be detailed in section 3.5. For now, we will only mention the AW state, already covered in the description of the other PIM protocols. The definition of these states is:

• A non-root interface can be in two states regarding the downstream interest on an (S, G) tree: DOWNSTREAM INTERESTED (DI) or NOT DOWNSTREAM INTERESTED (NDI). It is in the DOWNSTREAM INTERESTED state if it is connected to at least one interested neighbor or host and in the NOT DOWNSTREAM INTERESTED state otherwise.

• A non-root interface can be in two states regarding the forwarding state on an (S, G) tree: FORWARDING or PRUNED. The forwarding state of a non-root interface depends on its Assert state. It is FORWARDING if it is the AW and is PRUNED otherwise. Non-root interfaces directly connected to the source have to be always in the PRUNED state to avoid traffic loops.

• A router can be in two states regarding the interest on an (S, G) tree: INTERESTED (I) or NOT INTERESTED (NI). A router is INTERESTED if it has at least one non-root interface in the FORWARDING state and is NOT INTERESTED otherwise.

When a router is in one of the two states regarding the interest on an (S, G) tree, events may occur, and the adequate actions must be taken so that the network stays in a correct state. The events to which there must be a reaction are the reception of Join/Prune messages, the reception of the interest of hosts for a new tree, neighbor addition/removal, and interface role changes (caused by RPC changes). The detailed state machine of the interest of routers is described in appendix A (this includes the events that can occur in each state, the actions taken, and the state changes that can be triggered).

3.5 HPIM-SSM Assert

The Assert is used in point-to-point and shared links to elect an AW. The AW is the router responsible for forwarding the traffic to the link as well as propagating the interest towards the source. Without the election of the AW, if a downstream router sent a Join message to a shared link, every router would receive it and propagate this interest towards the root of the tree, creating parallel paths and duplication of traffic.

The AW is the non-root interface connected to the link that has the lowest RPC value to the source (in case of a tie, the interface with the highest IP address wins), thereby assuring the creation of an optimal path between the hosts and the source. The Assert message is used to elect the AW of a link for a specific source and group and contains the RPC of the sending router to the multicast source. It is multicasted through non-root interfaces and includes the IP address of the source and multicast group. The Assert process developed has the following properties:

- It has 2 states: Assert Winner(AW) and Assert Loser(AL)
- Only the non-root interfaces have Assert state regarding an (S, G) tree
- A non-root interface in the DOWNSTREAM INTERESTED state can be in the AW or AL state
- A non-root interface in the NOT DOWNSTREAM INTERESTED state is always in the AL state

Relation Interest-Assert and Stored States

The AW must have updated interest information to decide if it can remain AW and act as the link forwarder. The interest information sent by a neighbor must always be stored by the receiving interface irrespective of the current interface type and state (root, non-root, AW, AL) since any interface can become an AW. The AW election is triggered by the arrival of a Join message or interested hosts. Therefore, when the AW is elected it already has interest information. There is then a coupling between the interest and assert states.

Initially, all non-root interfaces in the link in the DI state consider themselves being the AW and transmit an Assert message containing the RPC of their routers. The root interfaces can not have Assert state and so, do not send an Assert message. If a non-root interface receives interest for some tree (S, G) but has no route on its unicast routing table for the source S it sends an Assert message with infinite cost. All interfaces in the link (root and non-root) receive the Assert messages and save the

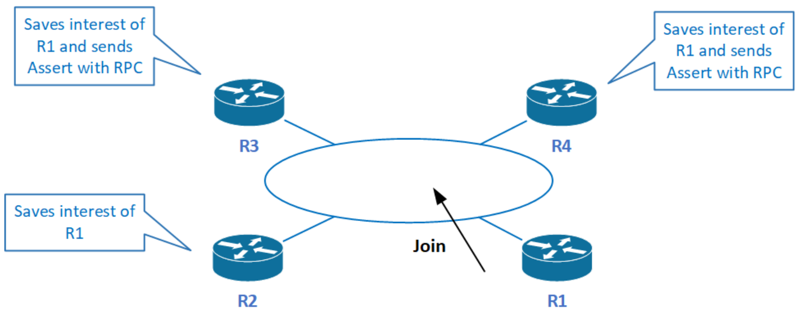

Saves interest of R1 and sends Assert with RPC
Saves interest of R1 and sends Assert with RPC
R3
R4
Saves interest of R1
Join
R2
R1

Figure 3.2: Interest and Assert

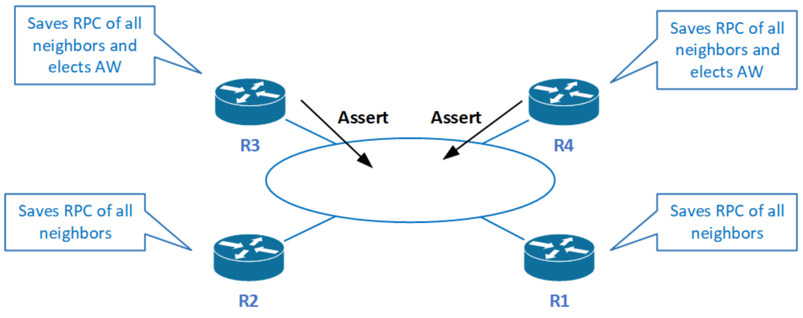

Saves RPC of all neighbors and elects AW
Saves RPC of all neighbors and elects AW
Assert
Assert
R3
R4
Saves RPC of all neighbors
Saves RPC of all neighbors
R2
R1

Figure 3.3: Assert process

RPC contained in them. The non-root interfaces will then use the RPCs stored to elect an AW. The root interfaces do not have an Assert state, but save the RPCs because they can become non-root interfaces and, therefore, potential AWs.

Figure 3.2 and Figure 3.3 show the trigger of the Assert and the Assert messages exchanged when R1 multicasts a Join message to the shared link. In both figures, R1 and R2 are connected to the link through a root interface and R3 and R4 through a non-root interface. All interfaces save the interest of R1 and the non-root interfaces of R3 and R4 send an Assert message with their RPC. All interfaces save the RPCs included in the Assert messages and the non-root interfaces of R3 and R4 elect an AW.

If there is a change of the RPC (without interface role change) in a non-root interface in the DI state (AW or AL), that interface has to send an Assert with its new RPC. Even if another router is elected AW, no additional messages need to be transmitted because interfaces know the more updated RPC of each other. If at a given moment there are no interested neighbors or hosts left for a specific (S, G) tree, the non-root interfaces will change from the DI to the NDI state and send an Assert Cancel message. An Assert Cancel consists of an Assert message but instead of containing the RPC of the router, it contains a predefined value (very high value), that acts as an infinite metric. It is used to alert the neighbors, the router can no longer be the one responsible for forwarding data in the link and its RPC can be removed. When an interface receives the Assert Cancel, it removes the RPC of the neighbor that sent the message. The other events that trigger the transmission of Assert Cancel messages are when an interface in the DI state (in the AW or AL state) becomes root or is removed.

When an interface is in one of the two states of the Assert, events may occur, and adequate actions must be taken so that the network stays in a correct state. The events to which the Assert must react are the reception of Assert messages, changes in the RPC of the router, neighbor addition/removal, failure of the current AW, and interface role changes. The detailed state machine of the Assert is described in appendix B (this includes the events that can occur in each state, the actions taken, and the state changes that can be triggered).

Initially, it was considered in our specification and implementation of HPIM-SSM an Assert similar to the one of PIM-SM/PIM-SSM protocols. This Assert had the same four properties mentioned above but routers only stored who the AW was and its RPC. The number of Assert messages needed to be transmitted was higher. We then identified some scenarios where different routers disagreed on who the AW was, although only one router considered itself the AW. We then developed a new Assert, the one we just described, that shows to be more efficient (less transmission of messages) and correct and with a simpler state machine.

3.6 Removal of tree state

The removal of the tree state is important, otherwise, routers would maintain indefinitely their multicast routing tables referring to (S, G) trees that do not exist anymore in the network. This state would only be removed by restarting the HPIM-SSM protocol in the routers. Removing the tree state is removing the entry in the multicast routing table regarding the (S, G) tree, including all downstream and the upstream interfaces associated with it. A router can remove the state regarding an (S, G) tree when all downstream interfaces are in the NDI state regarding that tree, and no assert information is stored. An interface has no assert information when no RPCs are stored of any neighbor. An interface removes the RPC of a neighbor when it receives an Assert Cancel message from it, when the neighbor fails, or when its Boot Time changes.

3.7 Installing and removing a tree

When a non-root interface receives the interest of a host or router referring to a new tree, it becomes DOWNSTREAM INTERESTED for that (S, G) tree. This will trigger the Assert, and if the interface becomes the AW, it changes to the FORWARDING state. The router then changes to the INTERESTED state since it has a non-root interface in the FORWARDING state and sends a Join (S, G) message through its root interface. The Join message will be received by the upstream routers, and the process is repeated until it reaches the first-hop routers. When a first-hop router is in the INTERESTED state, and the source is active, traffic is forwarded down the tree by the routers in the INTERESTED state until it reaches the interested hosts. Tree removal or pruning follows the same logic. When all non-root interfaces of the last-hop routers do not have any more hosts interested in receiving multicast traffic for a specific tree, they become NOT DOWNSTREAM INTERESTED, change to the NI state, and consequently to the PRUNED state. The routers then change to the NOT INTERESTED state since they no longer have non-root interfaces in the FORWARDING state and send a Prune (S, G) message through their root interfaces. The Prune message will be received by the upstream routers, and the process is repeated until it reaches the first-hop routers. When all routers are in the NOT INTERESTED state, it means that no hosts in the network are interested in receiving the multicast traffic for that source and the traffic is no longer forwarded. Like in PIM-SSM, trees can be built without the sources being active causing routers to consume computing resources unnecessarily.

3.8 Synchronization

Similarly to HPIM-DM, during the synchronization process, a new router learns information from its neighbors that allow it to integrate in the network correctly. In our case, since Join messages are not sent periodically, and the formation of the tree, as well as the Assert, is triggered by them, the new router must receive the equivalent of the information contained in the Join messages previously sent in the link, i.e. learn for which trees the routers connected to the link through a root interface (downstream routers of the link) are in the INTERESTED state. The new router must also learn the RPC of the routers that are connected to the link through a non-root interface in the DOWNSTREAM INTERESTED state since these routers sent an Assert with their RPC previously to the link. All routers in the link are already storing these RPCs to later perform an AW election locally when triggered. Remember that the local AW election does not trigger the transmission of Asserts even for the router that became AW.

The process that assures the consistency of synchronization and the synchronization protocol in HPIM-SSM is the same as in HPIM-DM and is described in section 2.2.5.I. The only thing that changes is the information exchanged during the synchronization process.

3.9 Message Sequencing and Message Reliability

The message sequencing and message reliability mechanisms are similar to the ones used in HPIM-DM and are described in sections 2.2.5.H and 2.2.5.J respectively.

3.10 Message Format

We present below the format of the protocol header that all control messages include as well as the specific fields of each one of the control messages. The format of all the control messages (including this protocol header) used in our implementation is identical to the ones used in HPIM-DM and can be accessed on the GitHub Repository [24] in the folder “docs/HPIMStateMachines.pdf”.

3.10.1 Protocol Header

The packet of all control messages includes this header that contains the following fields: BootTime, Version, Type, Security Identifier, Security Length, and Security Value. The definition of all fields except the Type is equal to the HPIM-DM protocol. The field Type determines the control message that is being transmitted and can have the following values: 1-Hello; 2-Sync; 3-Assert; 4-Join; 5-Prune; 6-Ack.

3.10.2 Hello, Assert, Join/Prune, ACK

The Hello and ACK message’s format is equal to the ones of HPIM-DM. The format of the Assert message including the definition of the fields is equal to the IamUpstream message of HPIM-DM. The Join and Prune messages have the same format in HPIM-SSM, having as the only difference the message’s type at the Protocol Header. Besides, this format is equal to the Interest messages’ format of HPIM-DM.

3.10.3 Sync

The Sync messages format is similar to the one of HPIM-DM. All fields are the same except the Sync Entry field. In HPIM-SSM, the Sync Entry field can have two different formats, depending on the information being exchanged. During the synchronization process, interest and assert information can be exchanged. To distinguish between these two types of information, every Sync Entry field includes an Identifier, that can have the value 0 or 1 if interest or assert is being exchanged respectively. When interest is being exchanged, the Sync Entry field has the following subfields: the Identifier, the Tree Source IP, and the Tree Group IP. When assert is being exchanged, the Sync Entry field has the following subfields: the Identifier, the Tree Source IP, the Tree Group IP, the RPC, and the RPC Preference. The RPC Preference is the preference value associated with the unicast protocol that provides the route to the source and the RPC is the cost to the source. For both cases, the Tree Source IP and Tree Group IP represent the IP of the source and IP of the multicast group of a certain tree (S, G).

Initially, it was considered another format for the Sync Entry field of the Sync messages that showed to be not correct in a specific scenario. In our implementation, the Sync Entry field has not been updated yet to the new format.

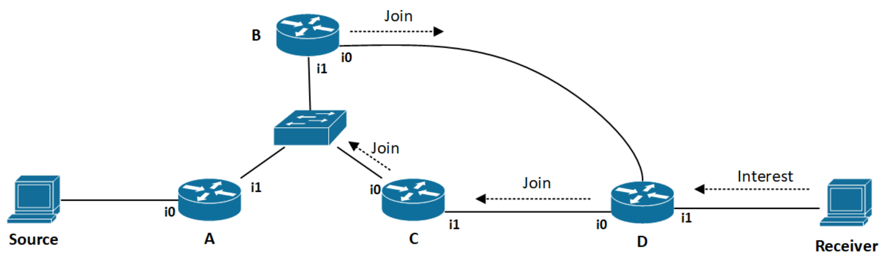


Figure 3.4: HPIM-SSM Loop Network

3.11 Existence of loops in HPIM-SSM

It can happen in a network with routers running HPIM-SSM, the creation of a loop. By loop, we mean that routers store tree state indefinitely, which can only be removed by restarting the protocol. A loop can only happen when the routers are running different unicast protocols. Figure 3.4 is an example of a network scenario that would create a loop. Every router except router B is running OSPF as the unicast routing protocol. Router B has a static route configured to D. Each router has a root and a non-root interface. The interfaces i0 are root and the interfaces i1 are non-root. Router D is notified that the receiver is interested in receiving multicast traffic from a certain (S, G) tree and sends a Join message through its root interface. Router C does the same and the Join is multicasted in the shared link triggering the Assert. Because router B has the static route, it wins the Assert. It sends a Join through its root interface (to router D), creating a loop. Router D does not send a Join message again because it was already in the INTERESTED state for that tree. If the source starts transmitting, the traffic will not reach the receiver since router A is not part of the tree.

This loop is a consequence of the obligation of an Assert process to decide who forwards traffic to the link and send Joins upwards. In fact, a router can be elected AW without being the next-hop router of any downstream neighbor in the link. In this example, the next-hop router of router C is router A and not router B.

3.12 HPIM-SSM Examples

In the network of Figure 3.5, there is one multicast source S1 transmitting for group G, two multicast receivers, four shared links (lk1, lk2, lk4, and lk6) and two point-to-point links (lk3 and lk5). There is a unicast routing protocol running (for example, OSPF). R1, R3, R4, and R5 have an RPC value of 10, and R2 has an RPC value of 15.

1.1) Source tree creation

The creation of an (S, G) tree is triggered by the manifestation of interest of the hosts. Suppose Receivers 1 and 2 become interested in receiving multicast traffic from source S1 and group G. A source

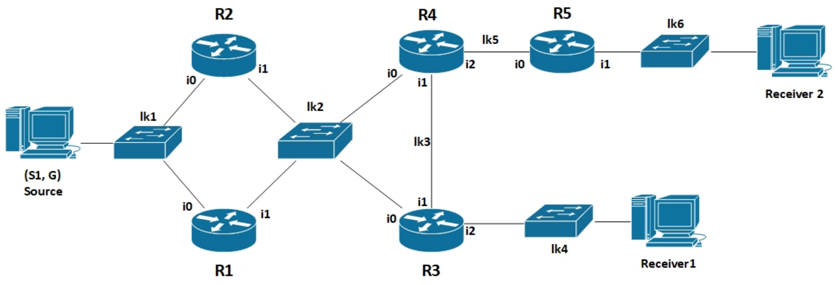


Figure 3.5: HPIM-SSM Network Topology

tree (S1, G) must be built. Receiver 1 signals R3 and Receiver 2 signals R5 using the IGMPv3 protocol. R3 and R5 save the interest of Receivers 1 and 2, respectively. Their non-root interfaces (i1-R5 and i2-R3) become DI, change from the AL to the AW state and become FORWARDING. The routers R3 and R5 change their state from NI to I and send a Join (S1, G) message through their root interfaces (i0-R3 and i0-R5). When the Join message is multicasted in the shared link (lk2) by R3, all the other routers in the link receive the message, triggering the Assert. A single forwarder is elected in the link, in this case R1, since it has the best RPC value towards the source. In the meanwhile, the interface i2-R4 received the Join from R5, saved the interest of R5 and become DI. Then, i2-R4 changes from the AL to the AW state and becomes FORWARDING. The router R4 changes its state from NI to I and also sends a Join (S1, G) message to the shared link lk2. Upon the reception of the Join message, the routers in the link save the interest of R4 in receiving traffic from the tree (S1, G). If the AW was already elected (because of the previous Join), the Assert would not be triggered again since both Join messages refer to the same tree (S1, G). All routers in the link have stored the RPC of R1 and R2 but only R1 and R2 have assert state, being R1 the AW and R2 the AL. The non-root interface of R2 (i2-R2) is in the AL state and thus in the NOT FORWARDING state (although being DI). R1 will be the router responsible for forwarding multicast traffic in the link lk2, R3 in the link lk4, and R5 in the lk6. The root interfaces of the routers in the I state (i0-R1, i0-R3, i0-R4, i0-R5) will receive the multicast traffic from the upstream neighbors, and the non-root interfaces in the FORWARDING state (i1-R1, i2-R3,i2-R4, i1-R5) will transmit the traffic downwards the tree.

1.2) Source tree removal

The removal process of an (S, G) source tree is similar to creation process. Suppose Receivers 1 and 2 stop being interested in receiving traffic from the tree (S1, G). Receiver 1 signals R3 and Receiver 2 signals R5 using the IGMPv3 protocol. The non-root interfaces of R3 and R5 (i2-R3 and i1-R5) remove the interest information stored relatively to the hosts, change to the NDI state and consequently to the PRUNED state. Both interfaces (i2-R3 and i1-R5) change to the AL state and delete the assert information stored. R3 and R5 change from the I to NI state since no non-root interface is in the FORWARDING state. This triggers the transmission of a Prune (S1, G) message through the root interfaces of both routers (i0- R3 and i0-R5). The non-root interface of R4 (i2-R4) becomes PRUNED (for the same reason as i1-R5), and the router changes to the NI state. Both R3 and R4 send a Prune (S1, G) message through their root interfaces to the shared link. All interfaces attached to the link remove the stored information regarding the interest of R3 and R4 in receiving multicast traffic from the tree (S1, G) and the ones that are non-root change to the NDI state. Interface i1-R1 changes from FORWARDING to PRUNED and the other non-root interface stay in the PRUNED state. Interface i1-R1 changes its assert state from AW to AL and interface i1-R2 stays AL. All remove the assert information stored. All routers are now in the NI state.

1.3) Synchronization

Suppose the tree (S1, G) from the source tree creation example described above is formed with R1 being the AW of the link lk2. If R1 reboots, all its neighbors will remove the previous information stored regarding R1 and then will synchronize with it. R2, upon the reception of the Hello message sent by R1 with the BootSN incremented, verifies the assert locally and considers itself the new AW (the RPC of R2 was removed previously). R3 and R4 save R2 as the new AW.

During synchronization, R1 will learn for which trees the routers connected to the link through a root interface are in the I state (R3 and R4). It will also learn the RPC of the routers connected to the link through a non-root interface in the DI state (R2). When R1 synchronizes with R2, it learns the RPC of R2. When R1 synchronizes with R3, it learns that R3 is interested in receiving traffic from the tree (S1, G) and stores this information. The same succeeds when R1 synchronizes with R4. After synchronizing with all its neighbors, the non-root interface of R1 (i1-R1) is in the DI state and in the AW state. Is in the AW state because is DI and has a better RPC than R2. R2 changed to the AL state after receiving the Assert message sent by R1, triggered when its non-root interface become DI (this behaviour is defined in the assert state machine in appendix B). Note that interface i1-R1 is in the FORWARDING state since it is both the AW and DOWNSTREAM INTERESTED.

2) Reaction to Network Events

We consider now only the shared link (lk2) of the network topology of fig. 3.5 and the routers attached to it (R1, R2, R3, R4). The non-root interface i1-R1 is in the AW and DI state and interface i2-R2 is in the AL and DI state.

2.1) RPC change without interface role change

Consider that the RPC of R1 changes without triggering an interface role change. Since its non-root interface is in the DI state, it sends an Assert message with its new RPC and all routers attached ti the link save this new RPC. If the RPC sent by R1 is better than the RPC of R2, the interface i2-R2 stays in the AL state, otherwise, it changes to the AW state. This behavior is described in the Assert state machine in appendix B.

2.2) RPC change with interface role change

Suppose the RPC of R1 changes causing its root interface (i0-R1) to become non-root and its non-root interface (i1-R1) to become root. When the interface i1-R1 becomes root, and since it is in the DI state, it sends an Assert with infinite cost. R2 upon the reception of this message, change to the AW state since it is now the only router that can Assert. Its non-rot interface is now in the FORWARDING state and so, the router changes to the I state and a Join is sent upwards. R1 changes its state from I to NI since it no longer has any non-root interface in the FORWARDING state, and sends a Prune message through its old root interface (i0-R1).

4

HPIM-SSM correctness tests

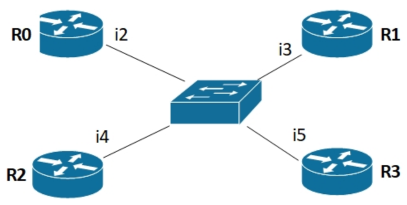


Figure 4.1: Model network topology

We describe here the model that was developed in the Promela language and tested with the SPIN tool [9] to verify the correctness of the HPIM-SSM protocol specification. More precisely, we want to prove that the state machines of the HPIM-SSM protocol are correct, so that regardless of the events that occur in the network and their order, the routers always converge to the correct state.

We modeled the assert and interest state machines in a single model to verify the correctness of each one as well as the combination of the two. The two state machines are highly linked, since the interest triggers the assert and the forwarding state of an interface depends on the assert and interest states. We did not model the synchronization process since it was already tested in a previous work (correctness of the HPIM-DM protocol [7]) and the process is the same in both protocols, having as only difference the type of information exchanged in the Sync messages.

The network topology used in the model is shown in Figure 4.1. This network includes:

- Four routers: R0, R1, R2, R3
- One shared link: connecting R0, R1, R2, R3
- R0 and R1 are connected to the link through their non-root interfaces (i2 and i3)
- R2 and R3 are connected to the link through their root interfaces (i4 and i5)

Below we describe the modeling of the assert together with the interest. To run the model the following commands must be executed:

```
spin -a FILENAME.pml
gcc -O2 -o pan pan.c -DVECTORSZ=15000
./pan -a
```

Interest–Assert Model

The code developed and used in this verification is in the file “interest_asssert.pml” and can be accessed in our GitHub repository [25] in the branch “Promela”.

In the model, each router stores all information in a global variable “node_info” and manipulates it by accessing it at an index represented by its router ID. The "node info" variable is a global variable of type NODE CONFIGURATION (typedef) and stores information related to the router: its RPC, its interest state (INTERESTED or NOT INTERESTED), and its interfaces. Each interface is represented by the variable "interface_configuration" of type INTERFACE CONFIGURATION (typedef) that stores information in relation to each of the interfaces of the router. The "interface_configuration" variable stores the type of interface, its assert state, the ID of the neighbor with the best RPC, the respective RPC of that neighbor, the interest of the interface (depends on the neighbor's interest), and its downstream interest (DOWNSTREAM INTERESTED or NOT DOWNSTREAM INTERESTED). The variable "neighbor_state" stores the interest of each neighbor (related with the reception of Join/Prune messages), and its RPC.

In total, four interfaces exist (two non-root and two root), connecting the routers to the shared link. Each interface has a channel associated with it, to exchange messages with its neighbors. Each message contains the type of message (Join, Prune, or Assert), the message source (interface ID), and the message destination (interface ID). These messages are multicasted, i.e., send to all channels of the neighboring interfaces. Each interface was modeled with two proctypes. Each proctype can be seen as an independent process in the model. One reads the messages from the channel and does the necessary state changes regarding the information on these messages (like saving the interest or RPC received from a neighbor, and verifying the possible changes in the interest and assert states). The other proctype checks the change in the interest of the router (INTERESTED or NOT INTERESTED) and transmits the respective Join/Prune messages. The change of interface roles, interface failure, and change in the RPC, was modeled by the execution of a specific function (different function for each event). The events and the respective actions implemented in this model are according to the specification. The verification of the convergence to the correct states was done through the LTL language (Linear temporal logic language). This language is used for verification in SPIN and allows us to define a correctness propriety of a system without being associated with any specific control points. We can then verify that something will eventually happen and this what we did in our model.

R0 and R1 were initiated with an RPC of 10 and R2 and R3 with an RPC of 20. The interest of R2 was set to INTERESTED to trigger the transmission of a Join message in the link and consequently the start of the interest and assert state machines in each router in the link. The non-root interfaces were initiated in the AL state. When an interface is non-root and becomes downstream interested it will consider itself being the AW and changes to the AW state. An interface changes to the AL state when stops being interested or it loses the assert battle. First, we tested this module without the trigger of other events that could change the network, like the failure of an interface or the change in the RPC of a router. In both cases, the routers always converge to the correct state, regardless of the order/time in which the assert state changes take place.

In this case, the non-root interface of R1 should end in the AW and DOWNSTREAM INTERESTED state and its router in the INTERESTED state. Remember that initially, both non-root interfaces of R0 and R1 consider themselves the AW, transmit an Assert message, and since they are DOWNSTREAM INTERESTED the router will change to the INTERESTED state. R1 is the router with the best RPC so the non-root interface of R0 will end in the AL state and the router in the NOT INTERESTED state. R3 is also in the NOT INTERESTED state. Additionally, the non-root interfaces of R0 and R1 and the root interface of R3 should be interested. This can be written in LTL language by:

```
ltl ltl_test {(<>([](ROUTER_INTEREST(0)==ni && ROUTER_INTEREST(1)==in &&
ROUTER_INTEREST(2)==in && ROUTER_INTEREST(3)==ni &&
DOWNSTREAM_INTEREST(0,0)==di && DOWNSTREAM_INTEREST(1,1)==di &&
INTERFACE_ASSERT_STATE(0,0)==al && INTERFACE_ASSERT_STATE(1,1)==aw &&
INTERFACE_INTEREST(0,0)==true && INTERFACE_INTEREST(1,1)==true &&
INTERFACE_INTEREST(3,3)==true)))}
```

We then tested concurrently with the scenario described above, the failure of R1 (with the failure of its non-root interface), the failure of R2 (with the failure of its root interface), the change of the interest of R3 to the INTERESTED state, and finally the change of the RPC of R0. We wanted to verify that eventually R0 and R3 would be in the INTERESTED state, and R1 and R2 would be in the NOT INTERESTED state. The non-root interface of R0 should be in the DOWNSTREAM INTERESTED state and be the AW. We also verified that the RPC of R0 is was correctly updated and that R3 had stored the new RPC of R0. This can be written in LTL language by:

```
ltl ltl_test {(<>([](ROUTER_INTEREST(0)==in && ROUTER_INTEREST(2)==ni &&
ROUTER_INTEREST(3)==in && ROUTER_INTEREST(1)==ni  &&
INTERFACE_ASSERT_STATE(0,0)==aw && DOWNSTREAM_INTEREST(0,0)==di &&
MY_RPC(0)==15 && INTERFACE_NEIGHBOR_METRIC(3,3,0)==15)))}
```

5 Protocol Implementations

Contents

In this section, we address the implementation of the HPIM-SSM and PIM-SSM protocols. Both protocols were implemented in Python3 and made to run in systems with the Linux operating system. Using Linux has several advantages, namely, the possibility of changing the multicast routing table and define the root and non-root interfaces for each multicast tree. We chose Python because it is a high-level language that allows us to abstract from the machine language. It has available high-level functions that implement multicast and networking operations that in other languages it would be much harder to implement and possibly with several limitations. Python has good readability and offers good documentation. It is an object-oriented language and that allowed us to build an implementation based on classes. Each class includes a set of methods and variables that perform very well-defined functions in the operation of the protocol. Classes are connected by association or inheritance.

This section is divided into two subsections: one addressing the implementation of HPIM-SSM and the other the implementation of PIM-SSM. The implementation of the HPIM-SSM and PIM-SSM protocols was based on the HPIM-DM [24] and PIM-DM [26] implementations respectively.The PIM-DM implementation is part of the previous work that developed the HPIM-DM protocol. The structure of the protocols, including the classes and their methods, is equivalent to the one of PIM-DM and HPIM-DM protocols and the state machines were adapted to the new protocols.

For both sections, there is a diagram of the protocol implementation and a brief description of its software architecture, including an explanation of each class. It is an objected-oriented implementation, where the rectangles represent the classes, the closed arrowheads represent association and open arrowheads represent inheritance. This object-oriented implementation including the associations and inheritance between classes is very similar to the one of the HPIM-DM and PIM-DM protocols.

5.1 HPIM-SSM

The implementation of the HPIM-SSM protocol was made according to its specification (chapter 3) and can be accessed in our GitHub repository [25]. It has the goal to demonstrate that the specification, our principle, is feasible and can function in real environments. Moreover, it would be not possible to perform the convergence time tests, that contribute to the convergence time analysis of the protocols, without an implementation. The IGMPv3 protocol was not implemented. Although the hosts must manifest their interest, all the other protocol functionalities could be implemented and tested, not being dependent on the IGMPv3. Figure 5.1 represents the HPIM-SSM implementation diagram. We now describe the function of each module in this diagram.

Run, Main: Run is the module closest to the user that allows him to interact with the protocol. This in-

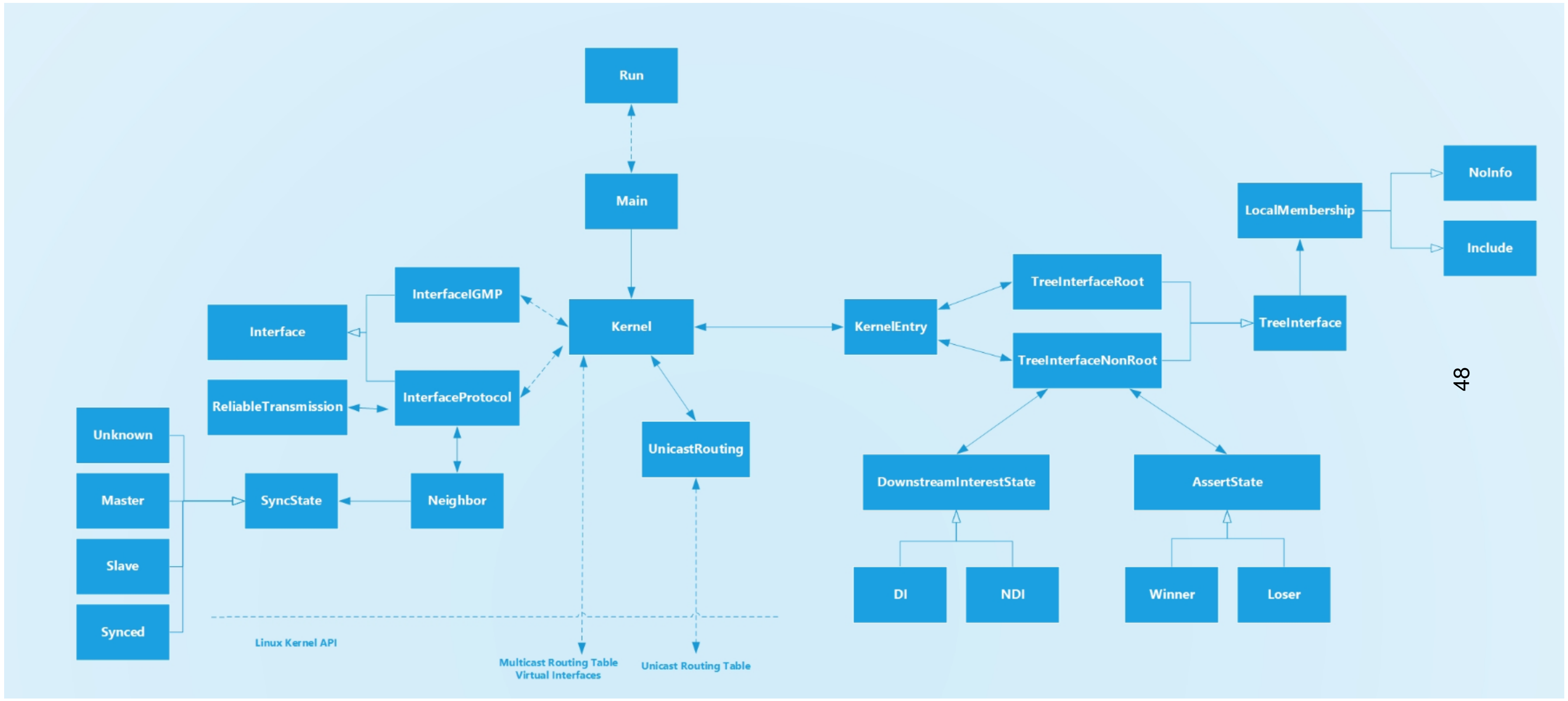


Figure 5.1: HPIM-SSM implementation diagram

teraction is through commands that the class receives and then interprets, calling the respective method in the Main module. The more important commands are the start and stopping the protocol in the routers and adding or removing HPIM-SSM interfaces. Main receives commands from Run and executes them in the Kernel module.

Interface, InterfaceIGMP, InterfaceProtocol: Interface is the super class of InterfaceIGMP and InterfaceProtocol and defines the methods that both subclasses must have (subclasses can differ in their implementation of the superclass methods). The InterfaceProtocol abstracts a physical interface where the HPIM-SSM protocol was enabled. It interacts with the ReliableTransmission and Neighbor classes. Its methods are related to the transmission and reception of the control messages (Hello, Assert, Join, Prune) and a thread with a configured socket is always running in the background, checking for the arrival of new messages. In HPIM-DM the InterfaceIGMP implements the IGMPv2 protocol (implementation developed in the previous work). The HPIM-SSM protocol only works with the IGMPv3 but this version was not implemented. Although not being used, to prepare future work and future integration of the IGMPv3 with the HPIM-SSM protocol, we included this class in the code with the IGMPv2 implementation.

Neighbor, ReliableTransmission: The Neighbor class represents an HPIM-SSM neighbor of an interface and stores its synchronization, interest, and assert state for each tree. It implements the four Sync states (Unknown, Master, Slave and Synced) and the Sync Timer. The interest and RPC are stored in two distinct dictionaries both having as key the tree (S, G) to which they refer to. The value of the dictionary that stores the interest of the neighbor is a Boolean that is TRUE when the neighbor is interested and false otherwise. The value of the dictionary that stores the assert, contains the more recent RPC received from the neighbor. If no RPC was received the value is empty. All sequence numbers known referring to that neighbor (BootTime, SnapshotSN, CheckpointSN, and messages SNs) are stored in this class. The ReliableTransmission class implements the ACK-protection mechanism for the control messages, namely, Assert, Join, and Prune messages assuring its reliable transmission. When one of these control messages is sent, a retransmission timer is started and the interface waits for all the acknowledges that should receive. It keeps track of the acknowledges received and if the timer expires without all neighbors acknowledging, the message is retransmitted.

Kernel, KernelEntry: The Kernel class is a bridge between the Linux Kernel and the protocol process. Its methods include creating and removing virtual interfaces (interfaces that have been enabled for HPIM-SSM) and changing the multicast routing table by creating, removing, or modifying entries. To perform these actions the protocol process connects and exchanges information with the Linux Kernel using a mroute socket. The Kernel class is associated with the KernelEntry class, which represents an entry in the multicast routing table. Each entry in the multicast table represents an (S, G) tree. Each KernelEntry instance can be associated with multiple TreeInterface classes.

TreeInterface, TreeInterfaceUpstream, TreeInterfaceDownstream: The TreeInterface class is a superclass of TreeInterfaceUpstream and TreeInterfaceDownstream. Each KernelEntry must be associated with one TreeInterfaceUpstream and one or more TreeInterfaceDownstream classes. The TreeInterfaceUpstream represents an interface considered to be root for a given (S, G) tree and implements the state machine of root interfaces. The TreeInterfaceDownstream represents an interface considered to be non-root for a given (S, G) tree and implements the state machine of non-root interfaces. It determines the Assert state (AW or AL) and the downstream interest state (DI or NDI) of the interface.

AssertState, Winner, Loser: The AssertState class defines the Winner and Loser states.

DownstreamInterestState, DI, NDI: The DownstreamInterestState class defines the DI and NDI interest states. The downstream interest state together with the Assert state determines the forwarding state of the interface.

LocalMembership, NoInfo, Include: The LocalMembership class implements the Local Membership state machine. It is associated with TreeInterface class. It knows if a given interface has members interested in receiving multicast data. This information together with the downstream interested neighbors determines the downstream interest state of the interface. In our case, since we are not using the IGMP protocol, the downstream interested neighbors determine alone the downstream interest of the interface.

UnicastRouting: The UnicastRouting class interacts with the unicast routing protocol. It sees the unicast routing table and detects changes in it. It uses the information in the unicast table to determine if an interface is root or non-root and the RPC of the router for each existing (S, G) tree.

5.2 PIM-SSM

The implementation of the PIM-SSM protocol was made according to the RFC7761 [5] and can be accessed in our GitHub repository [27]. The implementation was needed to perform the convergence time tests, and compare the times obtained with the ones of the HPIM-SSM protocol. Figure 5.2 represents the PIM-SSM implementation diagram. We now describe the function of each module in this diagram.

Due to the short time, we do not implement the designated router functions mentioned in the RFC7761. For the tests we performed and the events to which the time analysis was made, the designated routers' operations were not necessary to be implemented. The tests performed only involved the Join/Prune mechanisms for creating and removing trees.

Run, Main: The definition of the Run and Main modules is equal to the one of HPIM-SSM.

Interface, InterfaceIGMP, InterfacePIM: Interface is the superclass of InterfaceIGMP and InterfacePIM and defines the methods that both subclasses must have (subclasses can differ in their implementation of the superclass methods). The PIM-SSM protocol only works with the IGMPv3 but this version was not implemented. Nevertheless, to prepare future work and future integration of the IGMPv3 with the PIM-SSM protocol, we included this class in the code with the IGMPv2 implementation. The IGMPv2 implementation was developed in previous work.

The InterfacePIM abstracts a physical interface where the PIM-SSM protocol was enabled. It is associated with the Neighbor class. Its methods are related to the transmission and reception of the control messages (Hello, Assert, Join, Prune) and a thread with a configured socket is always running in the background, checking for the arrival of new messages.

Neighbor: The Neighbor class represents a PIM-SSM neighbor of an interface. It monitors the liveness of the neighbor and detects when it fails.

Kernel, KernelEntry: The definition of the Kernel and KernelEntry classes is equal to the one of HPIM-SSM.

TreeInterface, TreeInterfaceUpstream, TreeInterfaceDownstream: The TreeInterface class is a superclass of TreeInterfaceUpstream and TreeInterfaceDownstream. It defines the common methods used by both subclasses and is responsible for the assert state machine. Each KernelEntry must be associated with one TreeInterfaceUpstream and one or more TreeInterfaceDownstream classes. The TreeInterfaceUpstream represents an interface considered to be root for a given (S, G) tree. It stores the state regarding the upstream interface state machine including the respective Timers. A thread with a configured socket is always running in the background, checking for the arrival of new multicast traffic. The TreeInterfaceDownstream represents an interface considered to be non-root at a given (S, G) tree. It stores the state regarding the downstream interface state machine including the respective Timers.

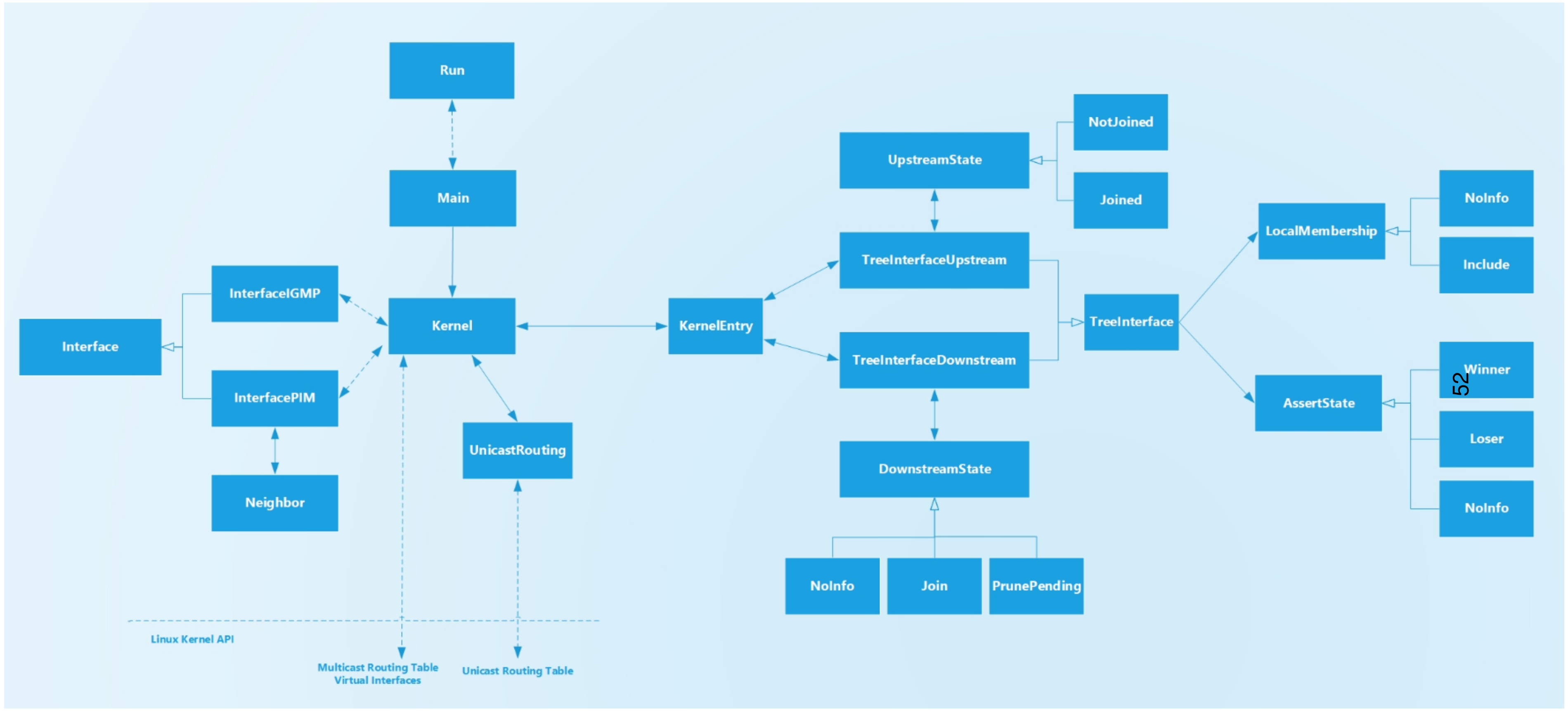


Figure 5.2: PIM-SSM implementation diagram

DownstreamState, NoInfo, Join, PrunePending: The DownstreamState class implements the Downstream Interface state machine. It defines the NoInfo, Join, and PrunePending downstream states.

UpstreamState, NotJoined, Joined: The UpstreamState class implements the Upstream Interface state machine. It defines the NotJoined and Joined upstream states.

AssertState, Winner, Loser, NoInfo: The AssertState class implements the Assert state machine. It defines the Winner, Loser and NoInfo assert states.

LocalMembership, NoInfo, Include: The definition of the LocalMembership class is equal to the one of HPIM-SSM.

UnicastRouting: The definition of the UnicastRouting class is equal to the one of HPIM-SSM.

6

HPIM-SSM implementation tests

Contents

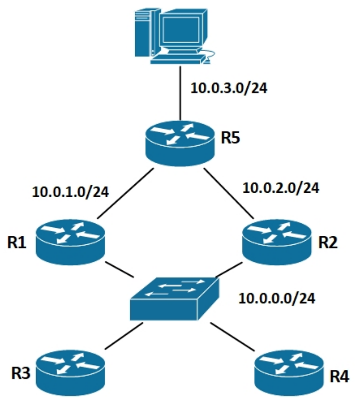


Figure 6.1: Network Topology

To verify that the protocol was implemented correctly we developed some tests in Python. We used the NetKit-NG software to perform the tests since it emulates a network environment and creates virtual machines. The routers need to run a unicast routing protocol. We chose the OSPF protocol which is being executed with Quagga [28].

All these tests were performed with the implementation of the protocol having the old assert. The synchronization process and its mechanisms are the same but the information being exchanged in the Sync messages was different (only interest information was exchanged). The new Assert and the additional information being exchanged in Sync messages were implemented after. Nevertheless, the correctness of the assert and its relation with the interest was verified with model checking techniques (chapter 4). The new Assert implementation, as well as the PIM-SSM implementation, were tested and corrected during the development of the convergence time tests. The convergence tests also have logs with all the changes in the state machines on the routers, namely, the transmission of control messages, the changes in the interest and assert machines. Before measuring the convergence of the protocols, we run the tests a few times and observe the logs to verify the routers were having the correct behavior. We verified the creation and removal of trees in both protocols. In HPIM-SSM we also verified the information exchanged during synchronization with a network with trees already formed as well as the reaction to network events like the change of interface roles.´

The network topology used was the same in all tests and is shown in Figure 7.2. This network includes:

- **Four subnets:** 10.0.3.0/24, 10.0.2.0/24, 10.0.1.0/24, and 10.0.0.0/24
- **Three point-to-point links:** Source-R5, R5-R1, and R5-R2
- **One shared link:** Connecting R1, R2, R3, and R4
- **All interfaces have a cost of 10 (OSPF)**

Apart from the network with the routers running the HPIM-SSM protocol we created a management network consisting of a central node (a router) connected to all other routers through point-to-point links (each one in a different subnet). This central node received logs from all routers containing the transmission and reception of control messages, as well as the state changes (assert, interest, neighborhood). All these events were registered in the logs with an associated time. To ensure all routers were synchronized we configured the NTP (Network Time Protocol) with the central node being the server and all other nodes the clients. We tested the synchronization with and without trees, the creation and removal of trees, and the forwarding of multicast data with an active source. Each one of these group of tests was performed in an automated way. For each group, we run a single time a python file in the central that performed all tests of that group. For each test, the central node initialized the routers (started the HPIM-SSM protocol, added the interfaces), put them in the initial state for the test, provoke changes in the network when necessary, and waited for them to be in the correct final states. Running a group of tests creates a single log containing information of all the tests. They are run sequentially (the next test only starts when the last one is successful) and that is how they appear in the logs. Each group of tests was run once, but the number of repetitions is a variable on the code that can be changed. Independently of the number of repetitions, we only have to run the python file in the central node once (for each group of tests).

6.1 Synchronization without trees

The synchronization without trees group of tests studies the creation and maintenance of neighborhood relationships between HPIM-SSM routers and consists of three distinct tests. The tests in this section are equal to the ones performed in HPIM-DM since both protocols have the same behavior regarding the initial synchronization of routers when no trees have been created yet. During this initial synchronization process, although routers do not exchange any tree information, they exchange Sync messages including the BootTime, SnapshotSN, and SyncSN. The transmission of Sync messages is protected using a Stop-and-Wait protocol, where one router is elected Master (the Master is responsible for controlling the communication process) and the other Slave. In the end, both routers must have their neighbor in the Sync state.

6.1.1 Test 1: Establishment of neighborhood relationship between two unknown routers

First, we started the HPIM-SSM process in all routers of the network. To have a clearer analysis and understanding of the events that happened in the network, we focus on R1 and R2. When the process is started, all the enable interfaces started transmitting Hello messages, containing the BootTime of the router. When R1 received the Hello message from R2, the synchronization process was triggered and R2 was elected Slave while R1 was elected Master. Both routers exchanged Sync messages containing theirs and the neighbor's BootTime and SnapshotSN as well as the SyncSN and the Master flag, indicating if the router that transmitted the message was the Master. In the end, we saw that both routers transited to the Synced state, knowing the correct BootTime of each other. The BootTime of R1 is 1590078525 and the BootTime of R2 is 1590078531.

6.1.2 Test 2: Re-establishment of neighborhood relationship after a router rebooting

Having as initial state, the final state of routers in Test 1, we rebooted R2, i.e., restarted the HPIM-SSM process. When the process is restarted, all the enable interfaces started transmitting Hello messages, containing the BootTime of the router. As expected, the BootTime of R2 (1590078581) is greater than before, while R1's BootTime stays the same (1590078525). After receiving the Hello message from R2, R1 initiated a new synchronization process electing itself as the Master and R2 as the Slave. The first Sync message transmitted by R1 has its BootTime and the neighbor's new BootTime. The SnapshotSN of R1 was set to 2 (bigger value than the last one) and the SyncSN was set to 0 since it is the first Sync message exchange in this synchronization process. Three more Sync messages are exchanged, being the next one transmitted by R2 to R1 as a response to the Sync message received with the SyncSN=0. The last two Sync messages have SyncSN set to 1 and the correct BootTime and SnapshotSN.

6.1.3 Test 3: Neighborhood relationship break after known neighbor failure

To test the tear down of a neighborhood relationship between two HPIM-SSM routers, we simulated the failure of R2 by stopping the HPIM-SSM process, which caused R2 to no longer transmit Hello messages. After a certain period of time without receiving Hello messages from R2, R1's neighbor liveness timer relatively to R2 expired and R1 removed R2 from its neighbors breaking the neighborhood relationship as expected.

6.2 Installing and removing a tree based on the Interest

This group of tests studies the states related with Interest namely, the downstream interest state, and the interest of routers in receiving multicast data. Although the Assert is dependent on the interest, we will focus our attention in the interest state transitions. Note that in HPIM-SSM, it is the interest of hosts and subsequently of the routers that triggers the creation and removal of a tree. In these tests, we created and then removed several trees, and analyzed that the routers had the correct state transitions. Because the creation and removal of a tree is triggered by the hosts, and since in the time of the execution of the tests, the IGMPv3 was not implemented yet, we programmed a router connected to the shared link through a root interface, R4, to multicast Join/Prune (S, G) messages to simulate the changes in the interest of the hosts..

6.2.1 Test 4: Creation of trees

In this test, we created five trees, and this was accomplished by having R4 sending five distinct and consecutive Join messages relatively to the following trees: (10.0.3.100, 224.12.12..12), (10.0.3.120, 224.12.12..12), (10.0.3.140, 224.12.12..12), (10.0.3.160, 224.12.12..12), and (10.0.3.180, 224.12.12..12). All the sources IP addresses are in the subnet 10.0.3.0/24, which is the network connecting R5 and the Source nodes (see network topology in Figure 1). Below we describe the creation of a single tree, but all the others followed exactly the same behavior. First, we verified that all routers in the shared link received the Join message, acknowledged it, and saved the interest of R4. Then we verified that the ones connected through a non-root interface, R1 and R2, changed their state to DOWNSTREAM INTERESTED (DI). The interest triggers the assert and R2 is elected the AW as expected. Since R2 had a non-root interface in the AW and DI state, i.e., forwarding, the router became INTERESTED (I), and sent a Join message through its root interface to R5. (Note: we can see in the logs that R1 also sent a Join and then a Prune message to R5. This happened because initially both R1 and R2 considered themselves the AW, changed to the (I) state and sent a Join upwards the tree, in direction to the source's subnet. After losing the assert, R1 changed to the NO INTERESTED (NI) state and sent a Prune message to R5). We then verified that the non-root interface connected to R2 was in the DI state and the one connected to R1 was in the NOT DOWNSTREAM INTERESTED (NDI) state. R5 did not send a Join though its root interface because it is directly connected through the source. At this point, all routers were in the correct state and the five trees were created successfully.

6.2.2 Test 5: Removal of trees

In this test, we removed the five trees created previously, and this was accomplished by having R4 sending five distinct and consecutive Prune messages relatively to the same trees mentioned above. Below we describe the removal of a single tree, but all the others' removal followed exactly the same behavior. First, we verified that all routers in the shared link received the Prune message, acknowledge it, and removed the previously saved interest of R4. As expected, the interfaces that were in the DI state (R1 and R2 non-root interfaces) changed to the NDI state. When this happened, R2 stopped having non-root interfaces in the forwarding state which made it change from I to the NI state. R2 then sent a Prune message to R5, that upon its reception, repeated the behavior of R2. The only difference is that R5 is directly connected to the source and so, it did not send a Prune when its state changes from I to NI. At this point, all routers were in the correct state and the five trees were removed successfully.

6.3 Synchronization with trees

This test studies the synchronization process when there are trees already formed in the network. Comparing with section 1.1, the routers must now exchange tree information in the Sync messages. Remembering the HPIM-SSM protocol, the new router must receive the equivalent of the information contained in the Join messages previously sent in the link, i.e., learn for which trees the routers connected to the link through a root interface (downstream routers of the link) are in the INTERESTED state.

6.3.1 Test 6: Discovery of tree information based on synchronization

To verify if the protocol implementation was correct, we started routers R5, R1 and R4 and created the five trees created in test 4 (using the same process). After the trees created, we started R2. The behavior expected in the network is to all routers synchronize with R2 and exchange tree information . After learning the existent trees in the network, R2 should become the AW since it is the router with the best RPC to the source. By checking the logs, we verified the routers had the correct behavior. R2 synchronized with R5, R4, and R1. Since R4 was the only router downstream in the INTERESTED state, it was the only that exchanged tree information with R2, namely the trees (S, G) to which is in the INTERESTED state (the 5 trees created before). After synchronizing with R4, the non-root interface of R2 became DI and considered itself the new AW (for all trees). Regarding the point-to-point link connecting R5 and R2, since R5 is upstream, there was no exchange of tree information during the synchronization.

6.4 Data Forwarding

The data forwarding group of tests verify that after the creation of trees, if a source becomes active, the multicast data is forwarded correctly down the trees and that if a source is active, but no trees exist in the network, no data is forwarded in the network. To verify the multicast data was being correctly forwarded, we configure the routers to send information to the logs every time a data packet was received in a root interface.

6.4.1 Test 7: Source is active and trees exist in the network

To verify if the protocol implementation was correct, we started all routers in the network and waited for all to synchronize with each other. We created the tree (10.0.3.100, 224.12.12.12) using the same process described in the tests before. R2 became the AW as expected. Then we activated the source and configured it to send a data packet every three seconds. R5, the router connected to the source received the packets and transmitted them only through its forwarding interface (the one connected to R2). R2 upon the reception of the data, repeated the behavior of R5, by transmitting the packet through its forwarding interface (the one connected to the shared link). Finally, R3 and R4 received the data. Note that R3 is not INTERESTED and so, in case of having downstream routers, it would not transmit the packets. On the other hand, R4 is INTERESTED and so, it would transmit the packets through its forwarding interfaces. The test was successful since the routers received the data through their root interfaces and the ones in the INTERESTED state, transmitted them through their forwarding interfaces (interfaces in AW + DI state).

6.4.2 Test 8: Source is active and no trees exist in the network

Having as initial state, the final state of routers in Test 7, we forced R4 to multicast a Prune message in the shared link to trigger the removal of the previously created tree. The behavior of the routers upon the reception of the Prune message was the one explained in Test 5. After the successful removal of the tree and with the source still active R5, the router directly connected to the source, received the data but did not forward it.

7 Protocols convergence time and stored state

Contents

7.1 Convergence Time

In this section, we compare the convergence time of the HPIM-SSM and PIM-SSM protocols. There were identified some events where the performance of the protocols is significantly different. We study the behavior of the protocols regarding each of the events by comparing the time they took to converge to the correct state. This section is divided into two subsections. In the first subsection, we address the theoretical convergence time values of both protocols regarding the events defined. We also explain how we reached those values. In the second subsection, we discuss the convergence time tests performed in protocol implementations to obtain practical values for the convergence time. We then compared the practical values with the theoretical ones.

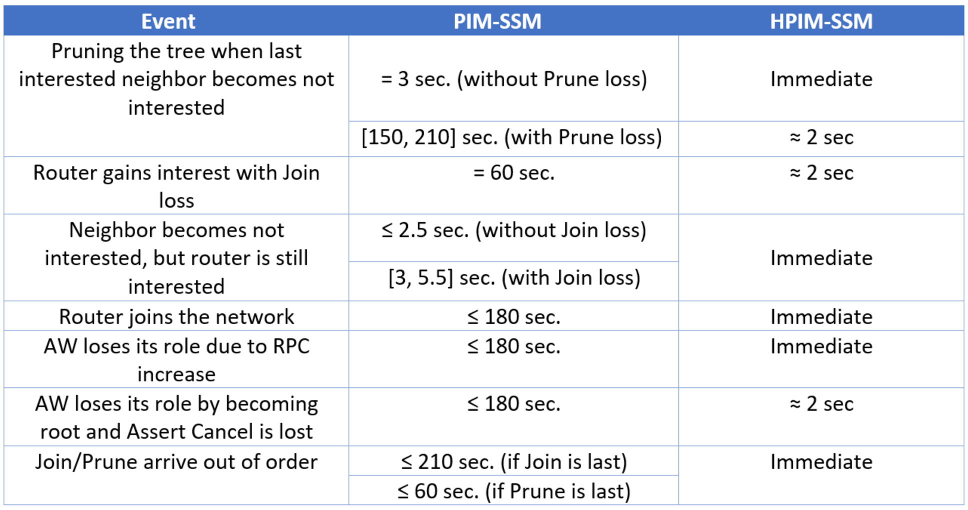
7.1.1 Convergence time: theoretical values

For each event, we calculated the theoretical upper limits of the time the protocols would take to converge to the correct final states. This information is summarized in Figure 7.1. The upper limits were calculated based on the RFC7761 [5] for PIM-SSM and based on our specification (section 3) for HPIM-SSM. We describe below each event, and how we arrived at the values presented in the table. We can easily observe that for HPIM-SSM all these events have almost an immediate reaction while in PIM-SSM values as high as 210 seconds can be reached. Although in the HPIM-SSM protocol messages can also be lost, the reliability of message transmissions ensures very fast recovery and convergence of the protocol. In the events with message loss, for the theoretical and practical values, we assumed only the first message was lost.

7.1.1.A Event: Pruning the tree when last interested neighbor becomes not interested

In PIM-SSM, if the last interested neighbor becomes not interested, it sends a Prune message upstream. If this Prune message is not lost, it takes 3 seconds (Prune Pending Timer) for the pruning to happen. In reality, the time it takes for removing the tree is a multiple of 3. In a scenario with chain links, it would take 3 seconds to prune each link, until reaching the source. If the Prune is lost, the problem is only corrected after the Expiry Timer expiring. The Expiry Timer has a value of 210 seconds and is set when a valid Join(S, G) is received. If no Join (S, G) message is received during this time, the timer expires and the interface goes to the NoInfo state. In the worst case, it could take up to 210 seconds for the link to pruned, when the Prune is sent immediately after the Join message. In the best case, the Prune is sent almost 60 seconds after the last Join message, and so, it takes near 150 seconds (210-60) for the link to be pruned.

In HPIM-SSM, control messages have a transmission reliability mechanism that allows the router that sent the message to know if it was received or not. Prune, Join and Assert messages use an



Event	PIM-SSM	HPIM-SSM
Pruning the tree when last interested neighbor becomes not interested	= 3 sec. (without Prune loss)	Immediate
	[150, 210] sec. (with Prune loss)	≈ 2 sec
Router gains interest with Join loss	= 60 sec.	≈ 2 sec
Neighbor becomes not interested, but router is still interested	≤ 2.5 sec. (without Join loss)	Immediate
	[3, 5.5] sec. (with Join loss)	
Router joins the network	≤ 180 sec.	Immediate
AW loses its role due to RPC increase	≤ 180 sec.	Immediate
AW loses its role by becoming root and Assert Cancel is lost	≤ 180 sec.	≈ 2 sec
Join/Prune arrive out of order	≤ 210 sec. (if Join is last)	Immediate
	≤ 60 sec. (if Prune is last)	

Figure 7.1: Convergence time values

ACK-protection mechanism and if the router that sent the message does not receive an ACK from each neighbor within a prespecified timeout period (two seconds), it retransmits the message and waits for the ACK again. If the Prune is lost, the router does not receive an ACK and retransmits the Prune after the 2 seconds. If the Prune is not lost, since there is no override mechanism and the upstream router knows it has only one downstream router still interested, the Prune happens immediately after the reception of the Prune from the last interested neighbor. (Note that when a router changes its interest state from INTERESTED to NOT INTERESTED it sends a Prune upstream right away).

7.1.1.B Event: Router gains interest with Join loss

In PIM-SSM, when a router gains interest, it sends a Join message upstream. If the Join is not lost, the upstream router is informed of this interest almost immediately. Otherwise, it takes 60 seconds, the time until the next Join message is sent (when a router is interested, Join messages are sent every 60 seconds, the default value of the Join Timer).

HPIM-SSM, when a router gains interest, it sends a message upstream just like in PIM-SSM, so the upstream router is informed of this interest almost immediately. But contrarily to PIM-SSM, there is no mechanism of periodic transmission of Join messages when the router is interested. If the Join is lost, the message transmission reliability takes action, and, when the router does not receive the ACK of the Join, it retransmits it after the retransmission timer expiring (2 seconds).

7.1.1.C Event: Neighbor becomes not interested, but router is still interested

In PIM-SSM, when a router receives a Prune but is still interested it has to send a Join to override the Prune. This Join is sent after a uniformly distributed time in the interval [0, 2.5] seconds. (Override Timer). So, if the Join is not lost, it takes up to 2.5 seconds. If the override Join is lost, after the prune timer expiring, the upstream router that is in the Prune Pending state, transitions to the NoInfo state and sends a PruneEcho message. The PruneEcho message has the purpose to make the override happen again in case of the override Join being lost locally in the shared link. This message is not reliably protected just as Join and Prunes. So, if the Join is lost it will take up to 5.5 seconds (3+2.5), the sum of the Prune Timer with the Override Timer (if the PruneEcho is not lost).

In HPIM-SSM, there is no override mechanism. When a neighbor becomes not interested, we do not have to send any message to reinstate our interest in the link. The upstream router has that information already stored and knows there is still at least another interested neighbor in the link.

7.1.1.D Event: Router joins the network

In PIM-SM, if a new router joins a shared link with a better RPC than the current upstream routers two possible scenarios may happen: 1) the current upstream routers have no assert state and 2) the current upstream routers have an Assert state. In the first scenario, the unicast table of the downstream routers changes (the next hop changes) and they send immediately a Join to the new router, and a Prune to the old upstream neighbor. In the second scenario, the downstream routers do not take action and continue sending Joins to the current AW. Only after the assert timer expiring in the upstream routers, the situation is corrected. The tree is rebuilt and the traffic starts flowing through the new router (the one with the best RPC). This could take up to 180 seconds, the default value of the Assert Timer.

In HPIM-SSM, there is a synchronization process when a router joins the network. In this process, the new router learns for which trees the downstream routers in the link are in the INTERESTED state and also the RPC of the upstream routers in the link that have their non-root interface in the DI state. When a new router joins a shared link with a better RPC than the current upstream routers and synchronizes with them, it realizes it has the best RPC and considers itself the new AW. It also could happen that this new router synchronizes first with one downstream router. In this case, its non-root interface would change from NDI to DI and would transmit an Assert message, making the routers converging to the correct Assert state. In both cases, the router would become the AW during the synchronization process with the routers in the link. The time it would take for rebuilding the tree after the new router joining the network, would be the time of the synchronization process, so almost immediately.

7.1.1.E Event: AW loses its role due to RPC increase

In PIM-SSM, if the AW suffers an RPC increase it should eventually become the AL. When the AW RPC increases, the downstream routers do not take action and continue sending Joins to the current AW. Only after the assert timer expiring in the upstream routers, the situation is corrected. The tree is rebuilt and the traffic starts flowing through the new AW. This could take up to 180 seconds, the default value of the Assert Timer. (In PIM-SM/PIM-SSM, this event with this convergence values, only happens when the Assert is active and a previous AW election occurred.

In HPIM-SSM, when the RPC of the AW increases, the router sends immediately an Assert message with its new RPC. All the routers receive this Assert, and the ones upstream in the DI state compare it with their RPC and the other RPCs they have stored from the other routers. The router with the best RPC will then consider itself the new AW.

7.1.1.F Event: AW loses its role by becoming root and Assert Cancel is lost

In PIM-SM, when the AW becomes root, it transitions to the NoInfo state and sends an Assert Cancel message with an infinite metric to the link. If this message is not lost, the routers in the AL state will consider themselves the new AW after receiving this message. If the Assert Cancel is lost, the downstream routers do not take action and continue sending Joins to the current AW. Only after the assert timer expiring in the upstream routers, the situation is corrected. The tree is rebuilt and the traffic starts flowing through the new AW. This could take up to 180 seconds, the default value of the Assert Timer.

In HPIM-SSM, when the AW becomes root, it sends an Assert Cancel message as in PIM-SSM. Since the Assert messages are protected with the ACK-mechanism, if the message is lost the ACK is not received by the old AW and the router retransmits the Assert Cancel message after 2 seconds.

7.1.1.G Event: Join/Prune arrive out of order

In PIM-SM, if a router sends a Join and then a Prune, but the Join is the last one to be received, the link will not be pruned as it should. The link will only be pruned when the Expiry Timer expires, which can take up to 210 seconds. On the other hand, if a router sends a Prune and then a Join, but the Prune is the last one to be received, the link can be pruned when it should not be. This problem takes up to 60 seconds to be corrected (Join Timer), the time until the next Join message is sent. The value of 60 seconds is only for links where the upstream router has only one PIM neighbor. Otherwise, a PruneEcho message is sent after the Prune Pending Timer expiring (3 seconds) and the problem is corrected up to a maximum of 5.5 seconds (Prune Timer + Override Timer) .

In HPIM-SSM, there is a message sequencing mechanism, to ensure messages are processed at the routers in the order they were transmitted. In this situation, the last message to arrive would have an SN lower than the first one to arrive, and so, the router would not take the action, since a message with a higher SN (containing fresher information) was already received and processed.

7.1.2 Convergence time: practical values

For the first three events of last section, we reinforced our confidence in the theoretical values obtained by performing convergence time tests. We did not perform these tests for the rest of the events due to their implementation complexity and not having the IGMPv3 implementation (applies for the fourth, fifth, and sixth events).

We used the implementations of the HPIM-SSM and PIM-SSM protocols to perform the tests. These tests simulate the events previously described, by creating an initial scenario in the network, triggering the event, and then measuring the time for the routers to converge to the correct final states. All the tests implemented can be accessed in our GitHub repositories: [27] for PIM-SSM and [25] for HPIM-SSM.

The network topology used was the same for all tests and is presented in Figure 7.2. All the tested events required triggering the creation of an (S, G) tree and for all tests, it was used the same pair (source, group): ('10.1.0.100', '224.12.12.12'). Since we do not have the IGMPv3 implementation, and the hosts are the ones responsible for triggering the creation of the trees, we had to trigger this creation in some other way. To do so, we forced the interest in R6 and/or R7 (when needed) and as a consequence, the transmission of Join(S, G) messages upstream. For simulating the no interest of the hosts, the same procedure was used. We forced R6 and/or R7 to become not interested and as a consequence, the transmission of a Prune (S, G) message upstream. This was done in both protocols.

Similarly to the implementation tests of HPIM-SSM described in chapter 6, we created a management network, with a central node connected to every router by a point-to-point link. The central node received logs from all routers containing all state changes and reception/transmission of messages. All these events were registered in the logs with a time associated. All routers had the NTP protocol configured. They were the NTP clients and the central node the NTP server. For the convergence time tests, is very important that all routers are synchronized in time, and with the NTP we assured that. The convergence times were calculated from the logs, by seeing the time difference between the initial and final state of routers. The initial and final states took into consideration for each test are indicated below in this section. Each event corresponds to one test, but there is a different test for the case of loss of messages. The tests are automated meaning we run the file of each test once, and the test was repeated the specified number of times, with each repetition of the test being associated with a different log. For example, a test

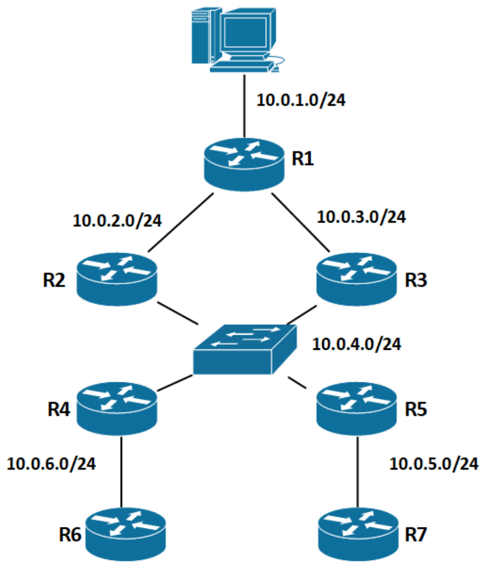


Figure 7.2: Network Topology

with 500 repetitions, was performed by running one time the python file in the central node that runs in an automated way each repetition sequentially (only after the last test being successful is run the next).

For the event *Pruning the tree when last interested neighbor becomes not interested without Prune loss* and the event *Router gains interest*, we only ran the respective test 10 times for the PIM-SM protocol, since these tests have a deterministic nature and as explained above, the time it takes for the protocol to converge to the correct state is always the same. The tests executed for these same two events for the HPIM-SSM protocol were run 500 times. The tests of all other events were run 500 times for both protocols. Below we explain with more detail how we performed each test and the results obtained.

We present now the results obtained in the convergence time tests for the three events. In Figure 7.3 are summarized the results for the HPIM-SSM and in Figure 7.4 for the PIM-SSM protocol. For both figures, the X-axis represents the events tested and the Y-axis the time in seconds the protocol took to converge. We can easily see that for the HPIM-SSM protocol all values are within the interval [0, 2.5] while in PIM-SSM we have a much larger interval of [0, 210] seconds.

7.1.2.A Event: Pruning the tree when last interested neighbor becomes not interested

To replicate this event in our network, the PIM-SSM/HPIM-SSM protocol (depending on the protocol being tested) was started in all routers, and all their interfaces were enabled. We then build an (S, G) tree by simulating the interest of R6, and after the tree was fully created, we simulated the no interest of R6. R6 sends a Prune upstream, R4 stops having downstream interested neighbors, changes its state regarding that tree, and sends a Prune upstream to R3.

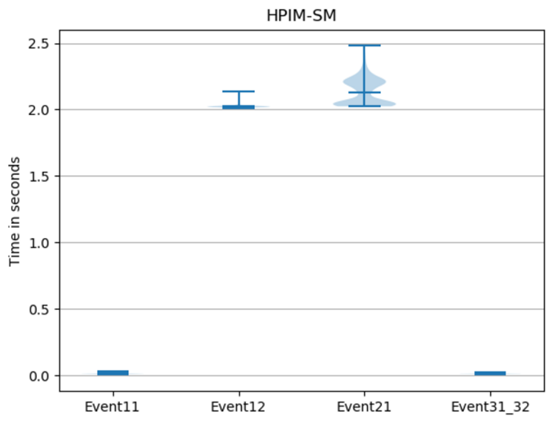


Figure 7.3: HPIM-SSM results

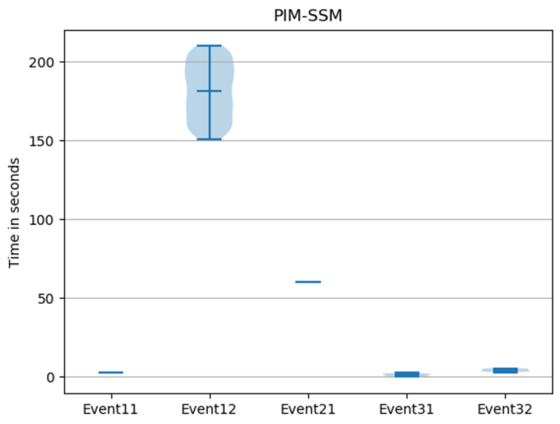


Figure 7.4: PIM-SSM results

For PIM-SSM, we measured the time since R4 changed its upstream state from Joined to NotJoined until R3 changed its downstream state to NoInfo. For HPIM-SSM, we measured the time since R4 becomes NOT INTERESTED until the downstream interface of R3 becomes NDI.

This event divides into two: when there is or not the loss of the Prune sent by R4. A test was created for each one of the cases. As said before, the test without the Prune loss in the PIM-SSM protocol was only run 10 times, since it is a deterministic test, and so, the time the protocol takes to converge to correct states is always the same.

The results obtained are according to the theoretical values previously calculated and are presented in Figure 7.3 and Figure 7.4. The labels Event11 and Event12 map to *Pruning the tree when the last interested neighbor becomes not interested without and with Prune loss*, respectively. For the Event11, HPIM-SSM had all values near 0 [0.005, 0.033] and PIM-SSM had values in the interval [3.009, 3.018]. These values are explained by the hard state nature of HPIM-SSM, namely, saving the interest state of the neighbors and thus not needing to use timers or the override mechanism. For the Event12, HPIM-SSM had values in the interval [2.013, 2.137] and PIM-SSM in the interval [150.562, 209.943]. There is a significant difference between the protocols in this event. While HPIM-SSM has a retransmission mechanism with a retransmission time period of 2 seconds, PIM-SSM has to wait for a 210 second timer to expire to be able to Prune the interface when the Prune is lost.

7.1.2.B Event: Router gains interest with Join loss

To replicate this event in our network, the PIM-SSM/HPIM-SSM protocol (depending on the protocol being tested) was started in all routers, and all their interfaces were enabled. We then triggered the creation of an (S, G) tree by simulating the interest of R6. The interest is propagated upwards the network in the direction of the source. When R4 receives the Join (S, G) message, changes its state regarding that tree and sends a Join (S, G) upstream to R3.

For PIM-SSM, we measured the time since R4 changes its upstream state from NotJoined to Joined until R3 changes its downstream state from NoInfo to Joined. What makes this test interesting is that there is the loss of the first Join sent by R4, otherwise, the convergence to the correct states would be immediate. For HPIM-SSM, we measured the time since R4 becomes INTERESTED until the downstream interface of R3 become DI.

The results obtained are according to the theoretical values previously calculated and are presented in Figure 7.3 and Figure 7.4. The label Event21 maps to *Router gains interest with Join loss*. HPIM-SSM had values in the interval [2.024, 2.479] and PIM-SSM in the interval [60.216, 60.284]. There is a considerable difference in the results obtained. While HPIM-SSM has a retransmission mechanism with a retransmission time period of 2 seconds, PIM-SSM has to rely on the periodic transmission of the Join messages with a periodicity of 60 seconds to receive the Join that was lost.

7.1.2.C Event: Neighbor becomes not interested, but router is still interested

To replicate this event in our network, the PIM-SSM/HPIM-SSM protocol (depending on the protocol being tested) was started in all routers, and all their interfaces were enabled. We then built an (S, G) tree by simulating the interest of R6 and R7, and after the tree being completely formed, we simulated the no interest of R6. R6 sends a Prune upstream, R4 stops having downstream interested neighbors, changes its state regarding that tree, and sends a Prune upstream to R3.

For PIM-SSM, we measured the time since R4 changed its upstream state from Joined to NotJoined until R3 changed its downstream state from PrunePending to Join (PrunePending $\rightarrow$ Join without Join loss and PrunePending $\rightarrow$ NoInfo $\rightarrow$ Join with Join loss). For HPIM-SSM, we measured the time since R4 becomes NOT INTERESTED until the downstream interface of R3 checked it was still DI. After receiving the Prune, the interface updates the interest of R3, checks the interest stored for the other neighbors, and verifies that R5 is still interested.

For PIM-SSM, this event divides into two: when there is or not the loss of the override Join sent by R5. When R4 sends the Prune to R3, R5 is still interested and so, it will send an override Join to R3. For HPIM-SSM this scenario does not happen since there is no override mechanism. Given that, the test with the Join loss could only be executed on the PIM-SSM protocol.

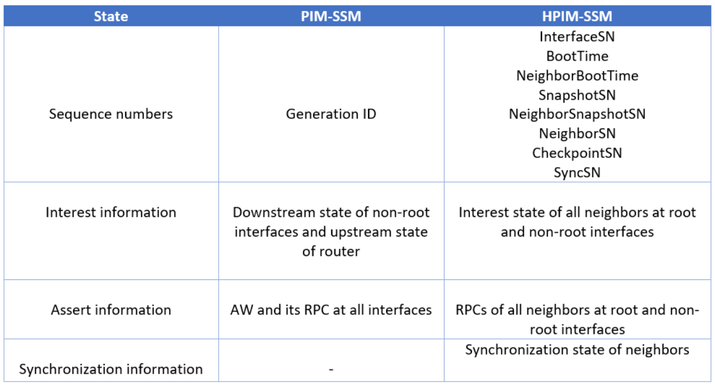
The results obtained are according to the theoretical values previously calculated and are presented in Figure 7.3 and Figure 7.4. The labels Event31 and Event32 map to *Neighbor becomes not interested, but router is still interested without and with Join loss, respectively*. The Event31_32 maps to the same event but for HPIM-SSM. For the Event31 and Event32, PIM-SSM had values in the intervals [0.022, 2.519] and [3.037, 5.528] respectively. This is explained by the override and prune pending timers. A detailed explanation was already given in the theoretical values subsection. For the event31_32 HPIM-SSM had values in the interval [0.009, 0.021]. There is a significant difference in the convergence time of both protocols. This is explained by the hard state nature of HPIM-SSM, namely, saving the interest state of the neighbors and thus not needing to use timers or the override mechanism.

7.2 Stored State

In section 7.1, we compared the time PIM-SSM and HPIM-SSM take to converge and conclude the HPIM-SSM has much faster convergence and reaction to network events. In this section, we compare the amount of information that needs to be stored at the routers for both protocols. It is clear that HPIM-SSM requires more stare information but less timers than PIM-SSM. Figure 7.5 summarizes the states that need to be stored and Figure 7.6 the existent timers of PIM-SSM and HPIM-SSM protocols.

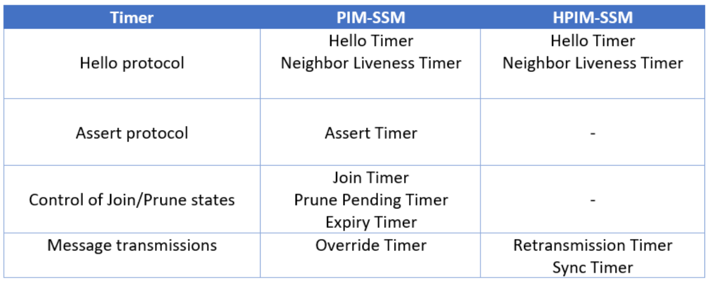
HPIM-SSM needs to store more state information regarding each neighbor than PIM-SSM. The sequencing and reliability of messages mechanisms, as well as the synchronization process, need to use several sequence numbers and some timers. All interfaces have to store tree state information regarding the neighbors i.e. assert and interest information. An interface has to store for each neighbor, tree state (its RPC and interest) and the synchronization state (UNKNOWN, MASTER, SLAVE, SYNCED). It also has to store sequence numbers regarding each neighbor that include the BootTime, the neighborSN, the CheckpointSN, the SyncSN, and the SnapshotSN. An interface with at least one neighbor needs to store its BootTime, interfaceSN, Hello Timer, and who its neighbors are. All interfaces need to store their type (root or non-root), and state regarding the ACK-protection mechanism (Retransmission Timer and pending acknowledgments). Regarding a tree (S, G), each interface saves its Assert, downstream interest and forwarding state. It also has to store state regarding the IGMP/MLD protocols to know if any local hosts are interested. Finally, each router needs to save the source and multicast group IP addresses of each existent tree, as well as its RPC and interest state (INTERESTED or NOT INTERESTED).

PIM-SSM does not store so many state information but uses more timers than HPIM-SSM. The Hello and Neighbor Liveness Timers are used by both protocols but PIM-SSM needs to use additional timers to control the Assert, the forwarding state of an interface and the pruning of links. PIM-SSM uses the following timers: Assert Timer, Prune Pending Timer, Join Timer, Override Timer, Expiry Timer, Hello Timer and Neighbor Liveness Timer. Regarding sequence numbers, each interface only needs to store the generation ID of each neighbor. Regarding the interest information, PIM-SSM needs to store the downstream state of non-root interfaces, but does not need to store this interest in root interfaces like it happens in HPIM-SSM. Regarding the Assert information, it needs to be stored in all interfaces who the



State	PIM-SSM	HPIM-SSM
Sequence numbers	Generation ID	InterfaceSN BootTime NeighborBootTime SnapshotSN NeighborSnapshotSN NeighborSN CheckpointSN SyncSN
Interest information	Downstream state of non-root interfaces and upstream state of router	Interest state of all neighbors at root and non-root interfaces
Assert information	AW and its RPC at all interfaces	RPCs of all neighbors at root and non-root interfaces
Synchronization information	-	Synchronization state of neighbors

Figure 7.5: Stored states



Timer	PIM-SSM	HPIM-SSM
Hello protocol	Hello Timer Neighbor Liveness Timer	Hello Timer Neighbor Liveness Timer
Assert protocol	Assert Timer	-
Control of Join/Prune states	Join Timer Prune Pending Timer Expiry Timer	-
Message transmissions	Override Timer	Retransmission Timer Sync Timer

Figure 7.6: Timers

AW is and the interface Assert state. Contrarily of HPIM-SSM, PIM-SSM only needs to store the RPC of the AW and not of all neighbors. Finally, the upstream state of the router (Joined or not Joined) has to be stored.

8 Conclusions

Contents

8.1 Conclusion

IP Multicast routing is essential for group communication applications since it helps saving network resources, being much more efficient and adequate than unicast or broadcast. PIM is one of the main multicast routing protocols with the sparse-mode protocols PIM-SM and PIM-SSM being the most popular and widely used. Its deployment is growing, and so, clients expect reduced traffic loss and fast convergence. Improvements made to the current protocol will lead to better performance and better service delivered by IP multicast applications. That is the motivation for our work. Some major limitations of the PIM protocols are slow convergence, slow tree reconfiguration, and network overhead caused by the periodic transmission of control messages and rebuild of the trees. There is also a lack of protection and ordering guarantees in the control messages. More specifically, in PIM-SSM since the trees are built from the bottom to the top and the DR can be the router not having the best RPC to the source, while the Assert is not triggered, the tree will not have the optimal path from the source to the interested hosts. It can happen that the assert is never triggered, or in the case that is triggered, it can happen a periodic reconstruction of the tree which creates instability in the network.

We developed a hard-state version of PIM-SSM, designated by HPIM-SSM, that has faster convergence and reacts to network changes almost immediately. These network changes are, for example, the appearance of a new router in the network, change in interface costs, change of interface roles, or rebuild of the tree due to change of interest in a router. Trees are built from the bottom to the top with an optimal path from the interested hosts to the source. In HPIM-SSM, the Assert is triggered by the interest and elects an AW in all links, as the interest is propagated towards the source. This ensures the creation of a tree with an optimal path from the hosts to the source and eliminates the need for the DRs. HPIM-SSM ensures that control messages are processed in the order they were transmitted and ensures the messages are received by all routers that should, due to a message sequencing and message reliability mechanism, respectively. Due to its hard-state nature, there is no periodic transmission of messages or periodic reconstruction of the trees. This reduces considerably the number of control messages transmitted in the network and reduces the data packets that could potentially be lost during these reconfigurations, contributing to reduced network overhead.

In this report, we presented a specification of the HPIM-SSM protocol that was then verified through model checking techniques. This verification is important to ensure that regardless of the network events happening in the network, or the order in which the routers in a link receive a certain control message, the assert and interest machines are always respected and that the routers always converge to the correct states. It was also verified the correctness in the interaction between the assert and interest state machines. The protocol was implemented in Python, based on the specification of HPIM-SSM, and tested using a network emulated environment. To compare the convergence time of the HPIM-SSM with PIM-SSM, it was also implemented in Python the PIM-SSM protocol. To evaluate our protocol and compare its convergence with the PIM-SSM protocol, we developed a set of convergence tests that were performed in our implementations of HPIM-SSM and PIM-SSM. The results obtained show that, for the events studied, the HPIM-SSM protocol has much faster convergence than PIM-SSM and the reaction is almost immediate. Also, significant time differences were noticed in the tests with loss of control messages, having HPIM-SSM a much better performance due to the reliable message transmission mechanism.

8.2 Future Work

Both protocols developed, HPIM-SSM and PIM-SSM, need to use the IGMPv3 for the hosts to manifest their interest in a specific source and group and so, an implementation of the IGMPv3 protocol, would be a very relevant complement to our work.

We implemented a hard-state version of PIM-SSM, which is a part of the PIM-SM protocol. A possible future work could be the implementation of a hard-state version of the entire PIM-SM protocol, including the creation of shared trees, the source registration, and switchover processes. One advantage PIM-SM has over PIM-SSM is that the shared tree is only created when a source is active, while in PIM-SSM trees can be created without the sources being active causing routers to consume computing resources unnecessarily. Just as HPIM-SSM, this hard-state version of PIM-SM would probably achieve good convergence results comparing with the PIM protocol.

The PIM-SSM and HPIM-SSM implementations were made for IPv4 multicast networks. Since the use of IPv6 networks is continually increasing, it could be interesting to develop an IPv6 implementation of both protocols. Additionally, for this IPv6 project, an implementation of the MLDv2 protocol could be developed. The IGMP is only used for IPv4 networks. For IPv6 networks is required the use of MLD and the functionality of IGMPv3 is implemented by MLDv2.

Bibliography

[1] B. Williamson, *Developing IP multicast networks*. Cisco Press, 2000, vol. 1.

[2] E. Rosenberg, *A Primer of Multicast Routing*. Springer, 2012.

[3] R. Wittmann and M. Zitterbart, *Multicast communication: Protocols, programming, & applications*. Elsevier, 2000.

[4] A. Adams, J. Nicholas, and W. Siadak, “Protocol independent multicast - dense mode (pim-dm): Protocol specification (revised),” Internet Requests for Comments, RFC Editor, RFC 3973, January 2005. [Online]. Available: https://tools.ietf.org/html/rfc3973

[5] B. Fenner, M. Handley, H. Holbrook, I. Kouvelas, R. Parekh, Z. Zhang, and L. Zheng, “Protocol independent multicast - sparse mode (pim-sm): Protocol specification (revised),” Internet Requests for Comments, RFC Editor, STD 83, March 2016. [Online]. Available: https://tools.ietf.org/html/rfc7761

[6] W. Odom, J. Geier, and N. Mehta, *CCIE Routing and Switching Official Exam Certification Guide (Exam Certification Guide)*. Cisco Press, 2006.

[7] P. F. C. de Oliveira, “Robust multicast routing protocol,” October 2018.

[8] Spinroot, “Promela manual pages,” last accessed 20 December 2019. [Online]. Available: http://spinroot.com/spin/Man/promela.html

[9] Spinroot.com, “Spin - formal verification,” last accessed 20 December 2019. [Online]. Available: http://spinroot.com/spin/whatispin.html

[10] Netkit-NG, “Netkit-ng homepage,” last accessed 20 December 2019. [Online]. Available: https://netkit-ng.github.io/

[11] Iana.org, “Ipv4 multicast address space registry,” last accessed 1 December 2019. [Online]. Available: https://www.iana.org/assignments/multicast-addresses/multicast-addresses.xhtml

[12] Iana, "Ipv6 multicast address space registry," last accessed 1 December 2019. [Online]. Available: https://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xhtml

[13] S. Deering, "Host extensions for ip multicasting," Internet Requests for Comments, RFC Editor, STD 5, August 1989. [Online]. Available: https://tools.ietf.org/html/rfc1112

[14] W. C. Fenner, "Internet group management protocol, version 2," Internet Requests for Comments, RFC Editor, RFC 2236, November 1997. [Online]. Available: https://tools.ietf.org/html/rfc2236

[15] W. F. S. Deering and B. Haberman, "Multicast listener discovery (mld) for ipv6," Internet Requests for Comments, RFC Editor, RFC 2710, October 1999. [Online]. Available: https://tools.ietf.org/html/rfc2710

[16] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, "Internet group management protocol, version 3," Internet Requests for Comments, RFC Editor, RFC 3376, October 2002. [Online]. Available: https://tools.ietf.org/html/rfc3376

[17] R. Vida and L. Costa, "Multicast listener discovery version 2 (mldv2) for ipv6," Internet Requests for Comments, RFC Editor, RFC 3810, June 2004. [Online]. Available: https://tools.ietf.org/html/rfc3810

[18] Cisco, "Ip multicast technology overview," October 2001, last accessed 2 December 2019. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/ip_multicast/White_papers/mcst_ovr.html

[19] D. Waitzman, C. Partridge, and S. Deering, "Distance vector multicast routing protocol," Internet Requests for Comments, RFC Editor, RFC 1075, November 1988. [Online]. Available: https://tools.ietf.org/html/rfc1075

[20] J. Moy, "Multicast extensions to ospf," Internet Requests for Comments, RFC Editor, RFC 1584, March 1994. [Online]. Available: https://tools.ietf.org/html/rfc1584

[21] A. Ballardie, "Core based trees (cbt version 2) multicast routing," Internet Requests for Comments, RFC Editor, RFC 2189, September 1997. [Online]. Available: https://tools.ietf.org/html/rfc2189

[22] M. Simões, "Gns3 pim-ssm assert study," last accessed 2 December 2020. [Online]. Available: https://github.com/Marga97/GNS3-PIMSM-Assert-DR

[23] J. Moy, "Ospf version 2," Internet Requests for Comments, RFC Editor, STD 54, April 1998. [Online]. Available: https://tools.ietf.org/html/rfc2328

[24] P. Oliveira, "Hpim-dm implementation," last accessed 20 December 2020. [Online]. Available: https://github.com/pedrofran12/hpim_dm.git

[25] M. Simões, “Hpim-sm implementation,” last accessed 2 December 2020. [Online]. Available: https://github.com/Marga97/hpim_sm.git

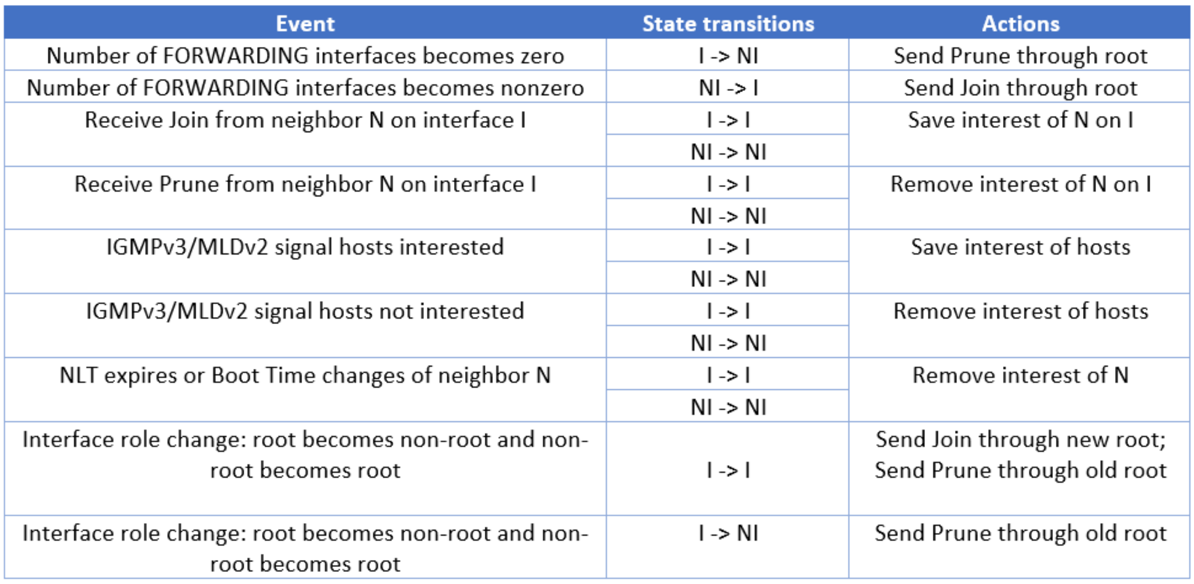
[26] P. Oliveira, “Pim-dm implementation,” last accessed 20 December 2020. [Online]. Available: https://github.com/pedrofran12/pim_dm.git

[27] M. Simões, “Pim-ssm implementation,” last accessed 2 December 2020. [Online]. Available: https://github.com/Marga97/pim_ssm.git

[28] P. Jakma, “Quagga routing suite,” last accessed 20 May 2020. [Online]. Available: https://www.quagga.net/

A

HPIM-SSM Interest State Machine



Event	State transitions	Actions
Number of FORWARDING interfaces becomes zero	I -> NI	Send Prune through root
Number of FORWARDING interfaces becomes nonzero	NI -> I	Send Join through root
Receive Join from neighbor N on interface I	I -> I	Save interest of N on I
	NI -> NI	
Receive Prune from neighbor N on interface I	I -> I	Remove interest of N on I
	NI -> NI	
IGMPv3/MLDv2 signal hosts interested	I -> I	Save interest of hosts
	NI -> NI	
IGMPv3/MLDv2 signal hosts not interested	I -> I	Remove interest of hosts
	NI -> NI	
NLT expires or Boot Time changes of neighbor N	I -> I	Remove interest of N
	NI -> NI	
Interface role change: root becomes non-root and non-root becomes root	I -> I	Send Join through new root; Send Prune through old root
Interface role change: root becomes non-root and non-root becomes root	I -> NI	Send Prune through old root

Figure A.1: (S, G) Interest State Machine

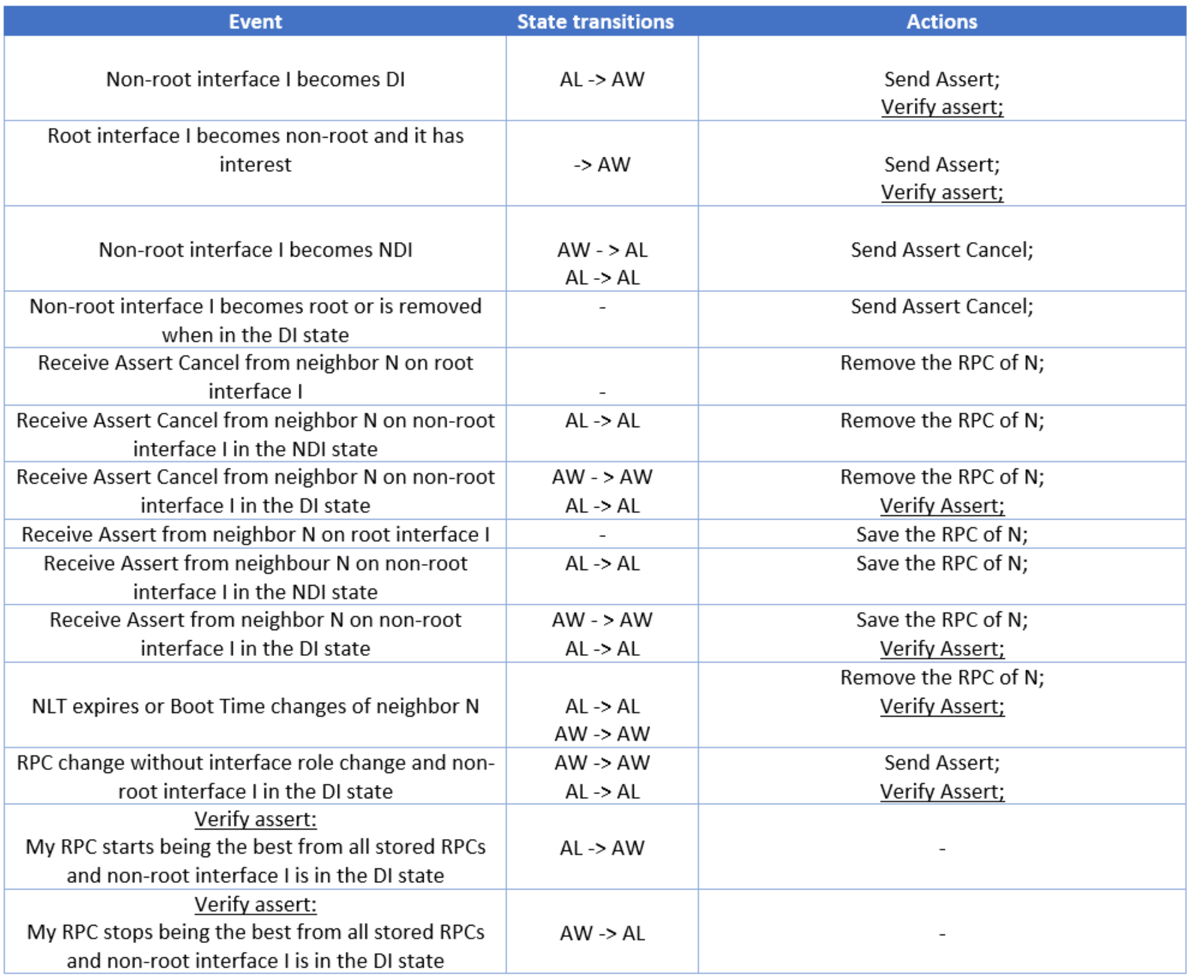
There are two states in the (S, G) Interest state machine:

INTEREST (I): This router has at least one non-root interface in the FORWARDING state for the tree (S, G). It is interested in receiving multicast data from the tree (S, G).

NOT INTERESTED (NI): This router has no non-root interfaces in the FORWARDING state for the tree (S, G). It is not interested in receiving multicast data from the tree (S, G).

B

HPIM-SSM Assert State Machine



Event	State transitions	Actions

Non-root interface I becomes DI	AL -> AW	Send Assert;
		Verify assert;

Root interface I becomes non-root and it has interest	-> AW	Send Assert;
		Verify assert;

Non-root interface I becomes NDI	AW - > AL AL -> AL	Send Assert Cancel;
Non-root interface I becomes root or is removed when in the DI state	-	Send Assert Cancel;
Receive Assert Cancel from neighbor N on root interface I	-	Remove the RPC of N;
Receive Assert Cancel from neighbor N on non-root interface I in the NDI state	AL -> AL	Remove the RPC of N;

Receive Assert Cancel from neighbor N on non-root interface I in the DI state	AW - > AW
	AL -> AL
		Remove the RPC of N;
		Verify Assert;

Receive Assert from neighbor N on root interface I	-	Save the RPC of N;
Receive Assert from neighbour N on non-root interface I in the NDI state	AL -> AL	Save the RPC of N;

Receive Assert from neighbor N on non-root interface I in the DI state	AW - > AW
	AL -> AL
		Save the RPC of N;
		Verify Assert;

NLT expires or Boot Time changes of neighbor N	AL -> AL
	AW -> AW
		Remove the RPC of N;
		Verify Assert;

RPC change without interface role change and non-root interface I in the DI state	AW -> AW
	AL -> AL
		Send Assert;
		Verify Assert;

Verify assert: My RPC starts being the best from all stored RPCs and non-root interface I is in the DI state	AL -> AW	-
Verify assert: My RPC stops being the best from all stored RPCs and non-root interface I is in the DI state	AW -> AL	-

Figure B.1: Assert state machine

There are two states in the (S, G) Assert state machine:

Assert Winner (AW): This router has won an (S, G) assert on interface I. It is now responsible for forwarding traffic from S destined for G out of interface I.

Assert Loser (AL): This router has lost an (S, G) assert on interface I or these interface is not interested. It must not forward traffic from S destined for G out of interface I.

The action Verify Assert consists in the interface comparing all the RPCs stored, including her own, and determine the router with the best RPC. After verifying the assert is possible that the interface: stays in the same Assert state, changes from AW to AL, or changes from the AL to the AW state.