Robust Multicast Routing Protocol

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

Instituto Superior Técnico, Lisboa, Portugal
October 2018

Abstract

Multicast routing protocols are an efficient solution for group communications, by providing logical connections between multicast sources and receivers, usually through one or more distribution trees, that minimize the bandwidth consumption and avoid delivering multicast traffic to network zones not interested in receiving it. Several multicast routing protocols were proposed for intra-domain routing, but the PIM protocols (PIM-DM and PIM-SM) are currently the preferred ones, being implemented by most vendors. PIM protocols are soft-state protocols that have several limitations, usually overcome through the periodic reconstruction of the multicast distribution trees. These limitations impair the use of these protocols in high-speed networks. This work proposes a hard-state version of PIM-DM. Unlike PIM-DM, the control messages are reliably delivered and sequenced according to their transmission order, and the construction of the multicast distribution tree is exclusively performed through control messages. Moreover, each router keeps track of all neighbors that can potentially become parents on the tree (i.e. to act as feeders of multicast data), a novel feature that allows keeping the tree correct even in the presence of network reconfigurations and failures. In addition, a synchronization process ensures that a new router attaching the network receives immediately the state information required to connect to the tree and start receiving multicast data. Finally, several optimizations were performed to minimize the amount of stored state and several features were included to avoid security attacks. The correctness of the protocol was assessed using logical reasoning and model checking, through the Promela language and the SPIN tool. The protocol was implemented in Python and was extensively tested in a network emulated environment.

Keywords: multicast routing protocol, multicast distribution trees, Dense Mode, Hard-State, HPIM-DM, PIM-DM.

1. Introduction

IP Multicast is a communication technique that allows a host to communicate with a group of hosts. This group can be on the subnetwork of the source or can span different subnetworks, being identified by a multicast IP address.

In IP multicast routing, routers create a logical structure, usually one or more distribution trees, for forwarding traffic from the multicast source to all interested receivers. For this to be possible, routers run a multicast routing protocol to create/remove trees and to trim existing trees, to remove redundant paths, and to attach new interested receivers and sources or remove no longer interested receivers and inactive sources.

Several IP multicast routing protocols have been proposed over the years. These protocols can be categorized as belonging to one of three categories, based on their working principles: Dense Mode, Sparse Mode and Link-State [21]. In Dense Mode protocols, such as PIM-DM [10] and DVMRP [20], routers first build a broadcast tree connecting the source to all routers, which is then pruned to remove redundant paths and paths that do not lead to interested receivers. In Sparse Mode protocols, such as PIM-SM [12], routers build a tree that is shared by the sources that transmit data to the same group, connecting all interested receivers. In Link-State protocols, such as MOSPF [15], a router has a global view of the location of sources and interested receivers.

PIM protocols (PIM-DM and PIM-SM) are currently the preferred protocols for intra-domain multicast routing, and are implemented by most vendors. These protocols keep being updated by the PIM working group of IETF [6]. However, they suffer from several limitations that compromise their performance. In this work, we have concentrated on PIM-DM, and specified and developed an hard-state version of this protocol that overcomes its limitations, called Hard-state PIM-DM (HPIM-DM). The correctness of the protocol was verified through logical reasoning and model checking, using the Promela [5] language and the SPIN tool [7]. The resulting specification was then implemented in Python.

Additionally, we have also implemented two other protocols in Python: IGMPv2 [13] and PIM-DM.
IGMPv2 allows multicast receivers to signal their interest in multicast traffic. PIM-DM was implemented in order to better understand how this protocol works, and also because there was no open implementation of its most recent specification. By implementing PIM-DM we also contributed to the open source and scientific communities.

All implemented protocols were tested in a network emulated environment, Netkit-NG [4], in order to verify if they were according to the specifications.

2. State of Art
Multicast traffic is characterized by a source that sends packets destined to a group of receivers. This group is identified by its multicast IP address, belonging to the subnet 224.0.0.0/4 in IPv4 networks [2] and FF00::* in IPv6 networks [3]. Hosts interested in a given multicast stream are called members. In the literature and in this document, for a certain multicast traffic, the notation S is used to refer its source IP address and G is used to refer its group address.

**Multicast distribution tree** When a host wants to send a multicast packet, it simply sends it (with the destination IP address set to the group’s address). For each multicast traffic, from a given source and destined to a given group, a multicast routing protocol will define a logical structure representing the paths through which multicast traffic should be forwarded. This structure is called multicast distribution tree and should only include paths that can reach its members. A multicast distribution tree has to adapt dynamically to the entrance and exit of its members and to possible failures of routers and links. This means that routers and links can be added or removed from the tree in order to deal with the problems previously mentioned.

The routers directly connected to the source are called originator routers. Note that there can be more than one originator for a given tree. The parent of a router is the router immediately above it in the tree from which multicast data must be received. The child of a router is a router immediately below it in the tree to which multicast data must be sent. Originators have no parent. The leaf routers of the tree are the routers that have no children.

All multicast routing protocols perform an incoming interface check as the primary mechanism to determine whether to forward or drop an incoming multicast packet. This verification is called Reverse Path Forward (RPF) [19, 11], since the forwarding is based whether the packet reached the interface that is on the “reverse path” back to the root of the tree. How a router defines which interface is on the “reverse path” is protocol dependent.

The interface of a router that receives multicast traffic from the parent router is called the root interface; the remaining interfaces are called non-root interfaces. The corresponding cost of a router to the root of the tree is called Root Path Cost (RPC), being this through the root interface of a router. Some non-root interfaces may not be allowed to transmit multicast traffic; this will happen whenever they have no downstream routers or receivers interested in receiving multicast traffic. Thus, a non-root interface can be in one of two states: FORWARDING, when allowed to transmit multicast data, or PRUNED, when not allowed to transmit multicast data. Under stable conditions, a router receives multicast data on its root interface, and retransmits it through its FORWARDING non-root interfaces.

Multicast distribution trees can be of two types: source trees and shared trees. The trees that routers form are protocol dependent.

- In source trees, the tree is rooted at the source of the multicast traffic (source S) and spans all members of group G, which belong to the leaves of the tree. The notation (S,G) is used to refer this type of trees. Source trees are usually formed by Dense Mode and Link-State protocols, like DVMRP, MOSPF and PIM-DM.

- In shared trees, the tree is rooted at some router at the core of the network, called rendezvous point (RP). In this type of structure, multicast traffic is forwarded from its source to the RP, and this router forwards multicast traffic down the shared tree to all its members. Routers that belong to the shared tree forward multicast traffic irrespective of its source. The notation (*,G) is used to refer this type of trees. Protocols that form shared trees are called Sparse Mode protocols, like PIM-SM.

**Membership protocol** Hosts manifest interest in multicast traffic to their directly connected routers by using a protocol independent from the multicast routing protocol: IGMP/MLD. With these protocols, routers are able to determine if there is interest in multicast data from directly connected hosts.

2.1. Overview of Dense Mode routing protocols
Dense mode protocols build source trees, delivering traffic from each active multicast source to a set of interested receivers, belonging to some multicast group. These trees are only maintained as long as the multicast source is active, i.e. keeps sending multicast data. This contrasts with unicast routing protocols, where the routing structure is maintained independently of the presence of unicast data.

Dense Mode multicast routing protocols start by building a broadcast tree, i.e. a tree that spans all routers. This is usually performed by flooding data packets throughout the whole network using RPF.

A critical issue in Dense Mode multicast routing protocols is to ensure that there are no redundant transmissions of multicast data, i.e. that data is not transmitted on a link without need. There are two cases: On point-to-point links, data must not be transmitted through both routers (both connected by non-root interfaces). On shared links, data must
be transmitted by only one of the non-root interfaces attached to the link. The protocol that selects the non-root forwarding interface of shared links is called the Assert protocol; this interface is called the Assert Winner (AW), and the remaining non-root interfaces are called Assert Losers (AL).

After removing redundant paths, we have a broadcast tree that still spans all routers. However, not all routers may be interested in receiving multicast traffic. This happens when a router has no multicast receivers attached to it and has no downstream routers interested in receiving multicast data. Routers not interested in receiving multicast traffic must be removed (pruned) from the tree. If later they become interested, they must be able to reattach to it. This is performed by the exchange of control messages, allowing routers to join and leave trees. These messages are propagated upwards in the tree, allowing to attach or remove paths that lead to these interested or not interested routers, respectively.

There are two main protocols implementing Dense Mode multicast routing: PIM-DM and DVMRP.

2.1.1 PIM-DM

PIM-DM [10] relies on a unicast routing protocol to determine the shortest paths from the routers to the multicast source. This means that the RPF technique takes into consideration the unicast routing table to select the root interface.

The construction of the tree is triggered by the first multicast data packet transmitted by the source. The packet is then transmitted using the RPF technique. Under this technique, an originator router transmits packets through all its non-root interfaces. Other routers, retransmit a packet received on a root interface through all their non-root interfaces, and discard packets received on non-root interfaces. This behavior causes an initial flood of multicast data packets, allowing all routers to know about the existence of the tree.

Initially we have all non-root interfaces forwarding data packets. The broadcast tree must be pruned in order to remove redundant paths and also to remove paths that do not lead to interested receivers.

To eliminate redundant non-root interfaces, there are specific techniques for point-to-point and shared links. On point-to-point links, when a non-root interface receives a multicast data packet (which it should not), it replies with a Prune message to the sending interface, which places that interface in PRUNED state. On shared links, when a non-root interface receives a multicast data packet, it replies with an Assert message, triggering the election of an AW. Assert messages carry the RPC of the sending router; the interface declaring the lowest RPC becomes the AW, and all remaining interfaces become ALs. Interfaces in AL state place their interface in a PRUNED state.

After removing all redundant non-root interfaces we have still a broadcast tree connecting all routers. Since not all routers may be interested in receiving data packets, these must be removed from it. This transforms the initial broadcast tree into a multicast distribution tree, that only spans interested receivers.

In order for a router to manifest its disinterest in a given tree, it transmits a Prune message as soon as it hears data packets from it. On point-to-point links, the reception of this message causes a interface to be placed in a PRUNED state. On shared links, the Prune Override mechanism is used, in order to control the forwarding behavior at the non-root interface of the AW. A Prune message is multicast so that all routers hear this message. If an interested router hears this message, it multicasts a Join message that has the goal of overriding the Prune. The AW by hearing a Prune message, waits some time in order to give opportunity to other routers to override the Prune with a Join. If the AW does not hear a Join, it places its interface in a PRUNED state, otherwise it places its interface in a FORWARDING state.

Links that were pruned from the distribution tree may later connect interested routers. In order for a router to connect again to a tree, it unicasts a Graft message destined to the AW. This message causes the receiving interface to acknowledge it, by transmitting a Graft-Ack message to the source of the first message, and the interface to be placed in FORWARDING state.

In PIM-DM, PRUNED state has a given lifetime. This means that a non-root interface placed in a PRUNED state will eventually timeout, transitioning to a FORWARDING state. This allows to deal with tree reconfigurations and possible inconsistent state at routers. This lifetime is of around 3 minutes, causing data packets to be forwarded periodically whenever this state expires. Later was introduced a mechanism, called State Refresh, that avoids the expiration of state, consisting in having the originator router periodically transmitting State Refresh messages, as long as the source remains active. These messages are forwarded down the tree by the AW at shared links and by non-root interfaces at point-to-point links, regardless of whether they are in a PRUNED or a FORWARDING state. A router by receiving this message, refreshes all its PRUNED interfaces. Besides this, State Refresh messages also include RPC information, allowing to reelect the AW.

2.1.2 PIM-DM issues

We have identified some issues present in PIM-DM that motivated the development of our protocol, HPIM-DM. PIM-DM has some issues that are only
solved by the periodic reconstruction of trees (can take at most 3 minutes) or by the periodic transmission of State Refresh messages (can take at most 1 minute). The State Refresh mechanism is an add-on and is not mandatory, i.e. may not be implemented at all routers.

Lack of reliability in Join and Prune messages Join and Prune messages are not reliably protected, causing the source of these messages to have no guarantee that these were successfully delivered. This can cause inconsistencies in the forwarding decision of routers. In shared links, by using the Prune Override mechanism, if the Join that is overriding the Prune is lost, the AW would stop forwarding data packets to the link. This issue is solved by the periodic reconstruction of the tree (at most 3 minutes) or by the forwarding of State Refresh messages (at most 1 minute).

If a link is pruned without being detected by an interested router it would be prevented from receiving multicast data.

Delay of Prune Override In shared links, when the AW hears a Prune it waits a given amount of time (default of 3 seconds) to allow routers to override it. This way of pruning is not efficient if there is no interested receiver and the topology consists in a chain of routers connected to each other by shared links.

New routers are unaware of previously formed trees New routers that attach to the network only discover new trees through the reception of corresponding data packets or State Refresh messages. If the router connects to a link that was previously pruned from the tree, it would only discover this tree after the non-root forwarding state expires (at most 3 minutes) or wait for the arrival of State Refresh messages (at most 1 minute). In the latter, the transmission of these messages by not being reliably protected, there is no guarantee that these would be correctly received.

Slow tree reconfigurations Trees are not reconfigured in reaction to RPC changes, causing these trees to have a sub-optimal configuration by not having data packets forwarding always through the best paths. This reconfiguration would happen eventually through the periodic reconstruction of the trees (at most 3 minute) or the exchange of State Refresh messages (at most 1 minute).

AW unaware of interested neighbors The router responsible for forwarding multicast traffic is unaware of which neighbors are interested, due to the Prune Override mechanism. If the only interested router fails, the AW would still forward data packets to a link. Eventually, other routers would transmit Prune messages causing the pruning of the link (at most 3 minutes without State Refresh or 1 minute otherwise).

Lack of message ordering guarantees PIM control messages are not sequenced and for this reason there is no ordering guarantees regarding the reception of messages. So if a router transmits two messages, the underlying network may change its order, not being detected at the receiver.

If a router transmits a Prune and then a Join/Graft and the underlying network changes its order, the AW would consider that its neighbor is not interested. This can cause the AW to prune prematurely a link, while there are still interested downstream receivers.

In case a router transmits a Graft/Join and then a Prune and the underlying network changes its order, this would be less problematic since data packets would still be forwarded. Nevertheless, if there are no interested receivers, these packets would be forwarded unnecessarily.

3. Hard-state Protocol Independent Multicast - Dense Mode
We have specified a new protocol, named Hard-state PIM-DM (HPIM-DM), that overcomes the issues identified at existing Dense Mode multicast routing protocols, such as PIM-DM.

3.1. Broadcast tree maintenance
Like any Dense Mode multicast routing protocol, a initial broadcast tree is formed, logically connecting any router to the source of multicast traffic. In our protocol a tree is built exclusively by control messages, named IamUpstream and IamNo LongerUpstream. These control messages are transmitted downwards the tree (through non-root interfaces) and its transmission is ACK-protected.

The IamUpstream indicates to a receiving router that the sending router can act as a feeder of multicast traffic, and the IamNo LongerUpstream indicates the opposite. The IamUpstream message also carries the RPC of the sending router to elect the AW and to eliminate routing loops.

Based on the received messages, routers store information on whether a neighbor is upstream or not: UPSTREAM and NOT UPSTREAM states. A neighbor is UPSTREAM if the last message received from it was an IamUpstream message. By storing information on all UPSTREAM neighbors, and not just on the parent router, is a distinguishable feature of this protocol that allows fast convergence when the tree changes.

The parent of a router in the broadcast tree is the UPSTREAM neighbor reachable through the root interface that provides the lowest RPC. Moreover, an Upstream neighbor is only considered parent if it provides an RPC lower than the router itself (i.e. closer to the source). This condition prevents the formation of loops. A router by having a parent on the tree, it transmits an IamUpstream downwards. Thus, a router by receiving an IamUpstream message it can be certain that all routers between itself
and the source form a tree structure that can forward multicast data.

By storing information about which neighbors are UPSTREAM, at non-root interfaces, allows them to determine if they are AW or AL. This storage at root interfaces allows them to determine which neighbor is the AW.

The storage of UPSTREAM neighbors and transmission of IamUpstream messages allows routers to form a broadcast tree. This process is initiated by the originator router when it starts receiving multicast data from a directly connected source. This router transmits IamUpstream messages through all non-root interfaces. Routers by having a parent become connected to the tree transmit IamUpstream messages downwards. Eventually any router will have a parent router, thus being connected to the tree.

When the source stops transmitting multicast data, the originator triggers the removal of the tree by transmitting IamNoLongerUpstream messages through all non-root interfaces. This process causes routers to stop having parents on the tree, thus transmitting IamNoLongerUpstream messages downwards. The tree is completely removed when no router connects to any neighbor UPSTREAM.

The process of removing trees is not exclusive to originator routers. A router by losing its parent (due to failure) becomes dis-attached from the tree, thus transmits an IamNoLongerUpstream messages downwards. The removal of a parent can cause all routers below it to no longer consider the tree as active or to consider other neighbors as parents, thus triggering the removal of tree or its reconfiguration.

Routers react to changes on the network, like RPC changes, router failures, removal or addition of Upstream neighbors, unicast changes by transmitting IamUpstream and IamNoLongerUpstream. Routers by notifying their neighbors about the corresponding upstream state allows routers to determine whether the tree is currently active or if it is being removed.

3.2. Interest maintenance
A router can be in one of two state regarding its interest on an (S,G) tree: INTERESTED, if the router is interested in receiving multicast data, or NOT INTERESTED, otherwise. This state depends on whether the router is required to forward data packets downwards the tree through one of its non-root interfaces, i.e. if it has at least one non-root interface in a FORWARDING state. An interface is in a FORWARDING state if it is both AW and connects to interested downstream devices (routers or multicast receivers); or in a PRUNED state otherwise.

The interest of multicast receivers is signaled through IGMP or MLD protocols. The interest of routers is signaled through Interest, NoInterest and IamUpstream messages.

A non-root interface by receiving an interest message is required to store the corresponding interest state of its neighbor. The storage of interest state of all neighbors allows the AW to make a correct and quick forwarding decisions, unlike in PIM-DM due to its Prune Override mechanism.

Multiple types of interfaces can connect a link: root and non-root interfaces. Non-root interfaces are not interested in receiving multicast data because these types of interfaces only forward packets, not being interested in receiving them. Interfaces that are root can be interested or not interested, according to the router's interest. So even if a router is interested in receiving multicast data, its non-root interfaces must manifest, to their links, that they are not interested in receiving data packets. A non-root interface by knowing the interest of all its neighbors can determine if there is downstream interest.

Interest and NoInterest messages can be transmitted (i) through root interfaces to manifest the router's interest, (ii) non-root interfaces to manifest the lack of interest. Since IamUpstream messages are transmitted exclusively by non-root interfaces, we also allow the interest state to be set using these messages (NOT INTERESTED state).

At a link, only one router must know the interest state of all its neighbors. This is the router responsible for making forwarding decisions to that link, i.e. AW. For this reason, interest messages are unicast to the AW, instead of being multicasted to all neighbors. This allows to reduce the number of exchanged messages, since these are reliably protected through an ACK-protection mechanism, so unicasting saves in the number of transmitted ACKs. On the other hand, the transmission of messages to a single router, causes interest state to be retransmitted whenever the AW changes. Additionally, in case a router detects that its interest is not correctly maintained at the current AW, it retransmits its interest to it. Since interest of routers change more frequently than changes at the AW, we opted to transmit these messages through unicast.

3.3. Message sequencing
Transmitted control messages need to be processed at intended receivers according to the order they were transmitted, otherwise the protocol may behave incorrectly.

We have devised a solution that: (i) sequence information is always monotonically increasing, (ii) transmitted messages use a single SN space, (iii) message sequencing is performed locally between neighbors, (iv) introduce an optimization that allows to periodically remove SN information.

In order to have ordering guarantees, messages carry two sequence numbers named BootTime and SN. The SN sequences messages for a given BootTime, such that whenever a new message needs to be transmitted, conveying more recent information,
its SN is incremented by one. The BootTime sequences reboots and SN overflows. The BootTime takes precedence over the SN in deciding if a message is fresher, thus a message with higher BootTime is always considered fresher. When two messages carry the same BootTime, the one with the highest SN is fresher. By having these two sequence numbers, control messages have always monotonically increasing sequence numbers, allowing to determine if they carry more recent information.

Routers transmit messages to set their state at its neighbors. Sequence information is per message type and tree, in the sense that neighbors must determine for each tree if received upstream and interest information is more recent than the ones previously stored. In order to reduce the amount of sequence information to be stored at routers, we have devised a solution that allows transmitters to use a single SN space for all trees and message types while receivers must only store SN information of their neighbors for each tree. This was possible by increasing the semantic scope of transmitted control messages, i.e. the same message can carry upstream and interest information.

Control messages by being used to set state at directly connected neighbors, message sequencing is performed locally at each link. For this reason, the router that transmits a message is responsible for correctly sequence messages and its directly connected receivers are responsible for correctly determining if received messages carry "fresher" state. By guaranteeing the local order of messages at a given link, this ensures that messages transmitted by any two pairs of routers (not directly connected) are also processed at the correct order, i.e. local order implies end-to-end order.

By using a single SN space for all types of messages and trees, this allows to reduce the overhead stored at routers. Nevertheless, each router must still store SN information for each neighbor and tree. In order to reduce even more this overhead, we have devised a solution called CheckpointSN, that allows routers to remove SN information of their neighbors, without causing any malfunction to the protocol. This consists in having a router notifying its neighbors about the lowest SN (CheckpointSN) to be transmitted in following messages. A router by receiving the CheckpointSN, stores this SN and removes all stored SNs lower than it. With this solution, a router can eventually store a single SN per neighbor, irrespective of the tree.

3.4. Synchronization

A router attaching the network must obtain information from its neighbors, to start receiving multicast data as soon as possible. In particular, it must learn which neighbors are UPSTREAM for each tree currently active in the network, so that it can connect to those trees. The synchronization besides allowing to synchronize state is also used to establish a neighborhood relationship.

A synchronization is triggered when there is the possibility of a neighbor to have a inconsistent state regarding a router, which can happen when: (i) new neighbor detected, (ii) known neighbor rebooted, (iii) bidirectional communication with known neighbor temporarily interrupted.

When two neighbors synchronize, they send to each other the list of trees for which they consider being UPSTREAM in relation to the neighbor, and the corresponding RPCs. Information regarding interest is not included in the synchronization process, since this information is only relevant to a single router (AW), being then transmitted through the usual interest messages destined to the AW of discovered trees.

A router by discovering UPSTREAM neighbors through the synchronization process can quickly be part of all broadcast trees that were previously formed. This information is exchanged in Sync messages.

Our synchronization process uses a procedure similar to the so-called database description process of OSPF [16], with some additional information, in order to synchronize state reliably.

A synchronization process must exchange information consistently. We guarantee this consistency by taking a snapshot of all trees that will be transmitted to the neighbor router as soon as this process is triggered. Additionally, we obtain the SN when the snapshot was taken, called SnapshotSN, that belongs to the SN space of the transmitting interface. All Sync messages will only include information from the obtained snapshot, even if tree information changes during an ongoing synchronization. This allows to guarantee that this information is consistently transmitted. Changes to this information are not included in Sync messages and are transmitted by the usual upstream and interest messages.

The SnapshotSN along with the BootTime of a router uniquely identify a synchronization period, being included on Sync messages. This means that all control messages transmitted prior to the synchronization have a SN lower than the SnapshotSN, thus being filtered. If the BootTime and SnapshotSN of both routers are included in all exchanged Sync messages, this allows a router to determine if a Sync message is referring to a current, previous or new synchronization process. Since the BootTime and SnapshotSN of both routers do not change in an ongoing synchronization, this allows routers to filter Sync messages from previous synchronization processes (lower sequence numbers), beside detecting when new synchronizations are required (higher sequence numbers).

Multiple Sync messages may be required to be exchanged, and like OSPF, routers elect a Master that exchanges the first Sync message and that controls
subsequent communications. The router that is not elected as Master is called Slave. This communication is protected using a Stop-and-Wait mechanism controlled by the Master.

3.5. Message reliability
All exchanged control messages are reliably protected, allowing a router to determine if the message has been delivered to all intended neighbors. By sequencing all messages, this allows routers to identify which messages have been reliably delivered.

The synchronization process uses a Stop-and-Wait protection to guarantee message reliability.

The remaining control messages: IamUpstream, IamNoLongerUpstream, Interest and NoInterest use an ACK-protection mechanism. In case of upstream messages, these are only considered delivered when all neighbors acknowledge them, while interest messages must be acknowledged by a single neighbor.

4. HPIM-DM correctness
In order to argue about the global correctness of HPIM-DM, we have subdivided it in multiple parts and argued about the correctness of each part individually. We have discussed about the correctness of the synchronization, tree maintenance, interest maintenance, message ordering and message reliability. Additionally, in order to support the correctness of the protocol, we have performed some formal tests through model checking.

In model checking, a model of the solution is created and a software tool is used to generate all possible states in which the program can be at and verify if correctness properties hold in those states.

SPIN [7, 14] was selected as the software tool used to perform those tests. In this tool, the model is specified in a programming language called Promela [5], which is similar to C. Then it is possible to verify if all correctness properties hold, by using linear temporal logic.

Linear temporal logic (LTL) allows to specify global correctness properties that are not associated with any specific control point of the model. By using LTL, it is possible to verify if a condition holds in all states of the model or if it will eventually hold.

4.1. Synchronization
The synchronization process in HPIM-DM is based on OSPF’s database description process [16], a known and well accepted synchronization mechanism. However, our synchronization process has some differences that offer even stronger guarantees regarding its correctness, by preventing replay attacks.

In HPIM-DM each synchronization process is identified by the BootTime and SnapshotSN of both routers. This allows routers to ignore messages that were replayed from previous synchronization processes. Basically we are using the synchronization process of OSPF and forcing both routers to exchange synchronization messages with this information. In case a router hears a message with a different BootTime or SnapshotSN (of itself or of its neighbor) the synchronization does not progress. This forces both routers to exchange information consistently since they can identify when both routers started the synchronization process.

In order to verify the correctness of the synchronization we have used model checking. We have modeled the synchronization process between two routers, through Promela. In this model we test if two routers can synchronize state consistently, requiring a random number of messages to be exchanged by both routers and in the presence of reboots and false suspect failures, through LTL at SPIN. This was proved in the developed model.

4.2. Creation and maintenance of trees
The maintenance of trees is correct due to the chain relationship of upstream routers. Routers by correctly notify their neighbors whether they are UPSTREAM or NOT UPSTREAM, allows them to determine if a tree still exists. This process takes into consideration the existence of network loops, changes at the unicast routing, changes of interface roles, failures and additions of routers, changes of upstream state at neighbors. A router reacts to these events by transmitting IamUpstream or IamNoLongerUpstream allowing the correct maintenance of upstream state at neighbors, causing the correct maintenance of trees.

In order to verify the correct maintenance of trees we have modeled the exchange of IamUpstream and IamNoLongerUpstream messages in a network topology, through Promela. In this model the originator triggers the creation of the tree by transmitting IamUpstream messages. These messages are forwarded down the tree causing all routers to discover and maintain this tree.

Routers store state regarding their neighbors and we test what happens if multiple routers fail concurrent to the construction of the tree. If there are redundant paths, routers must still maintain the tree state (and possibly change their root interface), otherwise they must stop maintaining tree state.

Additionally, we have verified if routers can create and remove trees in the presence of network loops.

All tests were verified using LTL at SPIN.

4.3. Interest
The maintenance of interest state was proved through logical reasoning. We argue that interest state is always or eventually stored correctly at the AW of a link. This is true due to the retransmission of interest messages in case routers suspect that the current AW no longer holds their most recent interest state. Also, in reaction to AW changes, routers retransmit their interest to the new AW.
By having routers correctly suspecting the loss of their interest state at the AW and by detecting an AW change, triggered by the reception of IamUpstream, IamNo Longer Upstream and failures of routers, it is possible to correctly maintain interest state.

4.4. Message sequencing
Regarding message sequencing, this is correct by having transmitted messages with monotonically increased sequencing information. Routers by storing the last SN regarding each tree and each neighbor can process messages at the right order, filtering messages carrying older state.

Several optimizations were performed without compromising its corrects. One of these optimizations was performing message sequencing per transmitting interface. This is correct by having enlarged the scope of transmitted control messages, i.e. a single message can carry interest and upstream state.

Other optimization was the CheckpointSN. Routers are able to remove SNs lower than the received CheckpointSN by knowing that following messages transmitted by that neighbor will carry necessarily higher SNs than the removed ones. By storing the CheckpointSN, this allows to filter retransmitted messages.

4.5. Message reliability
All upstream and interest messages are reliably protected by an ACK-protection mechanism. A message is only considered to be reliably delivered when all neighbors acknowledge a transmitted message.

Sync messages are reliably protected using a Stop-and-Wait protection. This is based on OSPF’s synchronization mechanism.

5. Implementation
In this work we have performed the implementation of three protocols: IGMPv2, PIM-DM and HPIM-DM. These implementations were developed in Python programming language and are targeted for Linux Operating System.

IGMPv2 was implemented in order for routers to detect the interest of multicast receivers. PIM-DM was implemented since there was no open implementation of its most recent specification, and also to learn how this protocol performs its duty regarding the forwarding of multicast traffic. HPIM-DM was implemented in order to have a working prototype of the specified protocol.

5.1. HPIM-DM
All communications between routers were performed through raw sockets. These used the Linux API in order to support the transmission and reception of multicast control messages.

Regarding the forwarding of multicast traffic this was accomplished by the Linux kernel, that already has a multicast routing table allowing a user-level process to manipulate its entries. Our implementation simply hears data messages, exchanges control messages and based on the internal state it sets and un-sets entries in the multicast routing table. A user level-process can define for each (S,G) tree, the root interface and all FORWARDING non-root interfaces (the ones that forward multicast traffic).

We have implemented a CLI interface for the user to interact with the protocol, so that it can add and remove HPIM-DM and IGMPv2 interfaces, and list all stored internal state (trees, neighbor state, sequence information, ...).

This implementation used some external libraries such as PyRoute2, PrettyTable, netifaces and ipaddress. The first one was used to obtain information from the unicast routing table (to obtain the RPC and also to select the root interface). The second one was used in the CLI interface, to return internal information in a human readable format (a table). Netifaces was used to obtain the IP address of a physical interface and ipaddress was used to perform calculations with IP addresses (verify if an IP it is inside a subnet and also to compare two addresses).

The implementation of HPIM-DM is stored in our GitHub repository here [17]. Our implementation of the latest specification of PIM-DM can be found here [18]. The implementation of IGMPv2 is integrated in both multicast routing protocols (in both repositories).

6. Tests
In order to check if the implementations were according to the specifications, some tests were done. All tests were performed using Netkit-NG [4], allowing to emulate a network environment. Additionally we have also used GNS3 [1] to verify the interoperability between our implementations executing in Linux devices and the implementations of Cisco routers, in case of IGMP and PIM-DM.

A test is defined by a certain topology in order to verify if a certain module of the protocol is working as expected. All messages exchanged between routers were captured using tcpdump [8] and further analyzed with Wireshark [9].

All tests were performed by knowing a priori the state in which some routers should be at. A centralized node knows this information and all routers send state transitions to it. A test is considered to be successful when all predicted state transitions were made. This was accomplished by sending logs of the implementation to a server node, according to state transitions, exchange of control messages and expiration of timers.

6.1. HPIM-DM
Regarding the tests to HPIM-DM implementation, these were subdivided into multiple modules in order to test each component of the protocol. We have performed tests regarding: (i) formation and maintenance of neighborhood relationships, (ii) creation
6.1.1 Formation and maintenance of neighborhood relationships

Table 1 lists all performed tests regarding neighborhood relationships. These tests were performed by having two routers connected to the same link. In these tests we have focused on the correct establishment and maintenance of neighborhood relationships in presence of reboots and fails, without having any tree involved in the synchronization process.

<table>
<thead>
<tr>
<th>Test Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Establishment of neighborhood relationship between two unknown neighbors</td>
</tr>
<tr>
<td>2 Re-establishment of neighborhood relationship after a known neighbor reboots</td>
</tr>
<tr>
<td>3 Neighborhood relationship break after known neighbor fails</td>
</tr>
</tbody>
</table>

Table 1: Establishment and maintenance of neighborhood relationships tests.

6.1.2 Creation and maintenance of the broadcast tree

Table 2 lists all performed tests to the maintenance of a broadcast tree. In these tests we have focused on the creation of the broadcast tree, by verifying if root interfaces were selected correctly, if neighbors transmitted expected messages and if the tree was being maintained correctly in the presence of network changes (RPC changes and failures).

<table>
<thead>
<tr>
<th>Test Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Initial creation of the broadcast tree (root interface selection and tree state)</td>
</tr>
<tr>
<td>5 Broadcast tree reconfiguration in reaction to RPC changes</td>
</tr>
<tr>
<td>6 Broadcast tree reconfiguration in reaction to router failure</td>
</tr>
</tbody>
</table>

Table 2: Broadcast tree formation tests.

6.1.3 Election of the router responsible for forwarding multicast traffic

Table 3 lists all tests regarding the election of the AW. In these tests we have focused if all routers elected correctly which interface was the one responsible for forwarding multicast traffic, in the presence of network changes and failures.

<table>
<thead>
<tr>
<th>Test Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Election of the AW in a shared link</td>
</tr>
<tr>
<td>8 Re-election of the AW in a shared link, in case the previous one fails</td>
</tr>
<tr>
<td>9 Re-election of the AW in a shared link, in case the previous one changes its interface role (its interface becomes root type)</td>
</tr>
</tbody>
</table>

Table 3: AW election tests.

6.1.4 Forwarding decision based on interest

Table 4 lists all tests regarding the forwarding decision based on interest of neighbors. In these tests we have focused on the forwarding decision of the router responsible for forwarding multicast traffic, by having multiple routers changing their interest.

<table>
<thead>
<tr>
<th>Test Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Forwarding decision based on neighbors’ interests, after the tree is initially built</td>
</tr>
<tr>
<td>11 A neighbor becomes not interested but there are still other neighbors interested</td>
</tr>
<tr>
<td>12 A neighbor becomes not interested and there are no interested neighbors</td>
</tr>
<tr>
<td>13 All neighbors were not interested and one them becomes interested</td>
</tr>
</tbody>
</table>

Table 4: Forwarding decision tests.

6.1.5 Discovery of tree through synchronization

Table 5 lists all tests regarding the discovery of trees through the synchronization process. In these tests we have focused on the synchronization process, that exchanged information to routers that have been previously connected to a link but that have restarted their interface.

<table>
<thead>
<tr>
<th>Test Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 Tree formation and correct multicast forwarding triggered by the synchronization process with a known neighbor (restart of non-root interface)</td>
</tr>
<tr>
<td>15 Tree formation and correct multicast forwarding triggered by the synchronization process with a known neighbor (restart of root interface)</td>
</tr>
</tbody>
</table>

Table 5: Discovery of trees through synchronization tests.

6.1.6 Maintenance of trees in the presence of loops

Table 6 lists all tests regarding the maintenance of trees in the presence of loops. In these tests we
have verified if a tree was correctly built and if it was then correctly removed, at all routers.

<table>
<thead>
<tr>
<th>Test Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16    Tree formation in the presence of network loops</td>
</tr>
<tr>
<td>17    Tree removal in the presence of network loops</td>
</tr>
</tbody>
</table>

Table 6: Creation and removal of a tree in the presence of loops tests.

7. Conclusions

From the analysis regarding existing IP multicast routing protocols, we have identified several issues that limit their correctness and convergence time. These issues can cause multicast distribution trees to have sub-optimal configurations, in case there are network changes. These protocols do not have any ordering guarantees, regarding transmitted messages, which can cause routers to perform wrong decisions in case messages arrive out of order. Also, not all messages are reliably protected and, therefore, can be lost and not delivered to their destinations. In order to overcome these issues, existing protocols use a soft-state approach and reconstruct the multicast distribution trees periodically, even in stable configurations, causing excessive control and data overhead.

In this work we have specified, implemented and tested a novel multicast routing protocol, which is a hard-state version of PIM-DM, aiming at overcoming its limitations. The protocol is named HPIM-DM (Hard-state PIM-DM). This protocol has distinguishing features that stand out from existing protocols, such as: maintenance of multicast distribution trees using an hard-state approach; routers maintain trees exclusively by control messages; the creation and removal of a tree is triggered by the originator router, but it can also be triggered by non-originator routers, in case they lose logical connectivity to the source of multicast traffic; the tree is reconfigured only in reaction to network changes, as soon as they are detected; message reliability and ordering guarantees; synchronization mechanism to exchange information regarding existing trees, allowing new routers to discover all previously formed trees as soon as possible.

The correctness of the specification was verified with logical reasoning and model checking, using SPIN. The synchronization and tree maintenance were verified with SPIN, confirming that all routers can reach a consistent state, even in the presence of router failures, network changes and network loops. The resulting specification was implemented with Python programming language and was extensively tested in a network emulated environment.

References