MEADcast: Explicit Multicast with Privacy Aspects

Vitalian Danciu*, Cuong Ngoc Tran*

* Ludwig-Maximilians-Universität München
Oettingenstr. 67, 80538 München, Germany
Email: {danciu, cuongtran}@mnm-team.org

***Abstract*—We find that existing multicast protocols require either the participation of hosts in group management or partial address lists of the group members to be sent to end-points (hosts), thus creating a privacy issue. Many services suitable for multicast are transmitted via massive unicast for technical or management reasons outside the sphere of influence of the sender. The large amount of identical payload transmitted constitutes a significant waste of network resources. We address these issues by presenting *MEADcast*, a multicast protocol intended to support a smooth transition from massive unicast to sender-centric multicast over the Internet. Senders perform all group management while receivers do not require explicit support for the protocol. The protocol copes with varying degrees of support by routers in the network and avoids the disclosure of end-point addresses to other end-points. Performance evaluation shows a decrease of the total traffic volume in the network of up to 1:5 as compared to unicast, suggesting suitability for applications, such as Internet Protocol Television (IP-TV), video conferences, online auctions and others.**

***Keywords*—Privacy-Preserving Multicast; Agnostic Destination; Explicit Multicast; Sender-Centric Multicast.**

I. Introduction

Applications replacing traditional broadcast services (IP-TV, IP-Radio), phone and video conferencing, and also technical services for software update or large-scale configuration may profit from an *n:m*, multicast, distribution scheme. Today, these applications still rely mostly on unicast transmission despite multicast having been available for a long time. In this text, we propose MEADcast (see also [1]), a multicast protocol intended to allow the optional and gradual introduction of 1:n communication between a sender and end-points into networks with initially unknown support for our protocol.

A. Challenges to Multicast Adoption

Many applications have evolved into their present form based on the technologies of the World Wide Web, including a connection-oriented, TCP-based communication layer and using the tacit design assumption that each service instance induces a one-to-one relationship between a service provider and a service user. The difficulty to roll out and maintain specialised client software for a service, compounded with the broad availability of a general-purpose, multimedia-capable client – the web browser – and the lack of pressure on service providers to conserve transmission capacity have led to the unicast design of applications that clearly lend themselves to multicast.

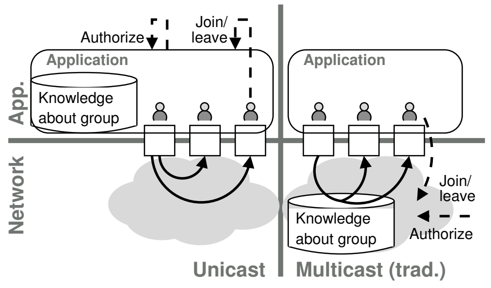


Figure 1. Knowledge and management actions in unicast and multicast.

B. Technical background

Typical multicast schemes are based on managed groups (e.g., [2], [3], [4]). End-points may join a multicast group and the network forwards messages addressed to that group to all its members, i.e., to all end-points that joined it. As a rule, a multicast group is symmetric in allowing any participant to address a message to all others. Unfortunately, it requires the network manager to effect configuration reflecting that a given application uses a different kind of network function, while the user is responsible for configuring the application to use multicast. The setup for services being provided across networks and thus across administrative domains always requires the cooperation of each participant domain's network managers.

Another important reason for the lack of multicast adoption seems to lie in the difference of scope of the application and the network function: the local scope of traditional multicast is inherent in the need to apply network configuration (e.g., IGMP) to the access network routers, while the applications are being provided from outside the local scope (i.e., the Autonomous System) over the global Internet.

As illustrated in Figure 1, by requiring an end-point to join and leave the multicast group that supports the desired application, the use of multicast

1) requires network management to authorize a service session and possibly setup (multicast routers),
2) requires the user to execute a network management action,
3) requires transfer of knowledge on group membership at the application level to a multicast group managed within the network layer and
4) introduces state to the otherwise state-less (from the view of the end-point) IP communication.

A number of additional properties exacerbate the perceived drawbacks to multicast use:

5) If the network-level setup of multicast fails, there is no automatic fall-back to unicast; instead, the application must detect and handle the failure.
6) All participants in a service must have multicast support.
7) Re-configuration of the application group requires re-configuration of the network.
8) Knowledge about the identities of the participants in a service session is present in the network, possibly in several administrative domains.

Applications seem therefore to prefer unicast even at the expense of the higher transmission volume, or Application-Layer Multicast (ALM) (e.g., [5], [6]) in spite of it being application specific and requiring a network function within the application's code.

In essence, ALM reduces the *n:m* multicast pattern to the asymmetric case of *1:n* communication, where a single sender addresses a group of receivers. In this case, it is sufficient for the sender to hold knowledge about the group. Since the sender necessarily implements the application layer of the service being provided, group management may be transacted at the application level. Such communication is easily implemented over unicast transmissions. However, it requires receiver-side configuration and does not profit from network support.

1) Non-proliferation of multicast: Although technical means to address inter-domain multicast continue to be developed (e.g., [7], [8], [9]), the adoption of multicast requires an incentive to both the service provider and to the access network operator. It requires the establishment of a cooperation between them by setting up routers supporting the inter-domain multicast scheme. What is more, it requires a mapping of services onto the (limited) multicast address space, implying a-priori knowledge about the services being used. Outside of applications provisioned centrally in the domain of an access network operator, this knowledge is generally not available: the reception of broadcast services (e.g., IP-TV, Internet radio) as well as the use of multi-party audio/video conferencing with participants outside of the network operator's domain is typically initiated by end users. Even when a certain service is provided locally, end users might still opt for an alternative service provided from outside the domain, if the local one requires a specific request or setup while the "foreign" one does not.

C. Contribution

We propose to combine the benefits of multicast to agnostic receivers with those of optional network support.

We introduce a protocol named Multicast to Explicit Agnostic Destinations (MEADcast) to allow sender-based multicast of IPv6 over the Internet. The novelty of MEADcast is that it protects receivers' anonymity and allows a gradual, pro-active and selective transition between multiple unicast and network-supported multicast. As the protocol favours conservative decisions, we present studies of the transmission cost in the network performed by simulating randomized as well as designed situations.

D. Technical overview

MEADcast implements a sender-centric multicast in that all knowledge about the receiver group, the network topology and the availability of MEADcast-capable routers (called *MEADcast routers* in this text) is gathered at and decided upon by the sender. Given an initial list of receivers, the sender commences to send data in unicast to each receiver while simultaneously probing the network for the presence of MEADcast routers and hence for the option to consolidate some of the unicast streams into multicast. Multicast packet headers reflect the MEADcast router responsible for translating the multicast packets into (multiple) unicast packets. Receiving end-points (called *receivers* in this text) always receive true unicast packets either directly from the sender or generated by a MEADcast router based on a multicast packet. Only unicast addresses are used in the protocol.

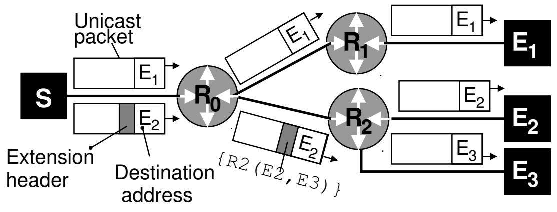


Figure 2. Multicast to agnostic receivers.

Multicast packets begin with a standard IPv6 header addressed to one of the multicast receivers on a path, followed by a Hop-by-Hop Routing Header with Router Alert. The addresses of all multicast receivers on a path as well as the MEADcast router(s) responsible for translation are encoded into a multicast header. It is typically followed by a UDP header. The addressing pattern is similar to the one in Internet email, where one receiver is addressed directly (To:) while all receivers are included in the carbon copy (CC:) list. The protocol is designed to minimize packet duplication, and receiver list re-writing in transit routers is eliminated.

Figure 2 shows an example where a sender S transmits to three receivers E_i with the aid of three MEADcast routers R_j. Note that the sender transmits unicast directly to E_1, as it is the only end-point on its subtree. It transmits one multicast packet to E_2 and E_3, to be transformed into unicast by router R_2.

None of the end-points can discern the identity of the others, thus preserving privacy, or if the data has been multicasted.

E. Synopsis

In the following Section II, we analyse the challenges to multicast adoption in dependency of the application being used. After describing a typical application scenario, we characterise applications with respect to their eligibility for multicast and discuss two example applications. Section III reviews different approaches to multicast, particularly Xcast [10], which is technically most related to our work, and juxtaposes their properties to those of our approach.

We continue by expounding the technical properties of MEADcast in Section IV by describing the behaviour of protocol entities, the structure of its protocol data unit and the API offered to applications. The description is complemented by a detailed example of the use of MEADcast. The study of the protocol's behaviour and performance, presented in Section V, indicates that the reduction in total volume may well be worth the effort for its introduction.

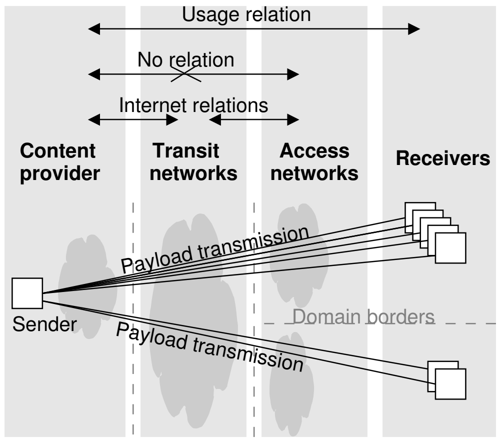


Figure 3. Relationships within the scenario.

In Section VI, we discuss the properties of MEADcast that address the challenges formulated in the Problem Analysis (Section II). In addition, the discussion addresses the protocol's overhead and limitations, security considerations and opportunities for optimization. Section VII summarizes our ideas and findings and points out further directions of research.

II. Problem Analysis

We analyse the problem of multicast adoption by characterising the applications that would benefit from it. The challenges faced by content providers are illustrated by means of a scenario. The analysis of applications' properties that influence their use of multicast are summarised in a problem space, which is exemplified by two different applications, one of them being the one from the scenario. The review in Section VI of the challenges identified in this section indicates how they are addressed by the MEADcast protocol proposed in this work.

A. Scenario

We illustrate the challenges to large-scale multicast use by means of the following scenario.

A media directory operates a website that lists freely accessible IP-radio and IP-TV offerings from different content providers, i.e., from Internet radio and TV "stations". The content streams are transmitted from the content providers to users in remote domains. Users that become increasingly aware of privacy issues are reluctant to use the service if other users become knowledgeable of their content consumption behaviour without their knowledge and consent. The privacy concept within our context is extensively discussed in the work "privacy terminology" [11].

Some content providers are willing to employ techniques like multicast in order to reduce the volume of data transmitted to the users. At the same time, some of the operators of networks hosting many receivers of the media streams are also interested in reducing the load within their own core and access network.

Both the media directory and the content providers are interested in reaching as many users as possible. Therefore, they are reluctant to create any barrier to the users' access of their content. The current and potential users are distributed globally over many domains, they have the skills to operate only basic, common software (e.g., web browsers) running on a diverse spectrum of types and versions of operating systems, thus obviating the effective introduction of a specialised client for receiving media streams.

Figure 3 illustrates the relationship between the content provider that hosts the sender, the transit networks and the networks hosting receivers. While there is a relationship of service delivery and service usage between the content providers and the users (brokered by the directory), this relationship is created ad-hoc based on interaction at the application layer. It does not extend to the network operators that host the users: the operators lack knowledge about the service being provided beyond what they may infer from observations of the network traffic entering their network. Content providers and network operators are associated with other networks by the relationships fundamental to the Internet, i.e., by peering and transit agreements. Such relationships are not necessarily between the two roles, and they are not transitive. Hence, in the general case, content provider and access network operator roles lack a direct relationship.

In consequence, it is difficult to establish a consensus for the use of multicast both technically due to the inter-domain aspect, and formally due to the lack of relationship between the content providers and the network operator hosting a user. There remains no other choice that is "safe" from preventing users' access to the offerings, than for the directory to broker unicast streams between the content providers and each of their users. This choice implies that a significant amount of the transmitted traffic will consist of identical payload.

B. Problem space

The scenario is exemplary of application scenarios that would benefit from multicast but do not use it. Such scenarios can be categorised by the properties of the applications arising from their purpose and their technical implementation:

- their communication pattern can be symmetric or asymmetric (i.e., m:n or 1:n communication),
- they can be operated within an administrative domain or across domains,
- their usage sessions can be of short or long duration,
- they can have a fixed or variable set of participants,
- they can have a small or large number of participants,
- they can have high or low demands on network resources,
- their capacity utilisation can be symmetric or asymmetric between participants, etc.

To be effective and efficient, the properties of the multicast scheme should be determined by the application category it serves.

The properties of the settings in which applications are deployed also characterise the scenario:

- network administration may be aware or unaware of the application,
- end-point users may or may not be allowed to self-configure group membership in a multicast group,

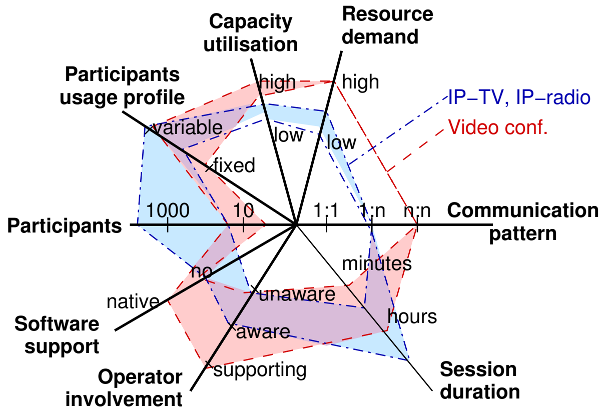


Figure 4. Application characterisation within a requirements space.

- the software package providing the application on the end-point side may or may not support group-based multicast.

C. Application examples

The resulting space is depicted in Figure 4, including two application scenarios described in the following, their usage requirements being shown as areas denoting their usage subspaces.

a) IP-radio and IP-TV: are part of a class of "broadcast" applications in that they replace traditional, e.g., terrestrial broadcast services. In analogy to the services it replaces, the application class exhibits a 1:n communication pattern and sessions that can reach durations of many hours, offered to a large, variable set of participants distributed throughout the Internet. The opportunity to employ buffering renders this application class insensitive to latency, jitter and drop rate and requires in principle only a sufficient end-to-end throughput according to the media transmitted. Network capacity is utilised in a highly asymmetric manner with a single sender producing all traffic once a session has been established.

b) Video conferencing: is an application with an n:n communication pattern of usage sessions in the range of a few minutes to a few hours between a small, variable set of participants that are grouped within a small number of domains. It has rather high requirements on quality of the network service due to being sensitive to latency, jitter, throughput and to some extent drop rate. The capacity utilised is symmetric, due to each participant sending his audio and video streams to all others. Hence, an application session creates a full-mesh network with traffic volume growing with the square of the number of participants. This may be the motivation for some operators to explicitly support multicast transport of conferences in order to reduce the peak throughput requirements on the network. Conferencing software typically supports the use of multicast addresses, however, group management, i.e., the creation of a group for a conference and the joining and leaving of multicast groups by participants, requires management information with respect to the mapping of conferences to multicast addresses. The participation of users outside the domain of the network operator cannot easily be supported from within the domain.

D. Challenges and opportunities

The challenges and opportunities in scenarios suitable for multicast can be summarised as follows.

- Both 1:n and m:n communication patterns offer a high degree of potential load reduction.
- The rather high volume requirements of pseudo-multicasted streams of media (e.g., audio, video) increase the potential reduction.
- The number of receivers of a particular content varies strongly, and receivers must be expected to enter and exit the group at any time.
- The duration of a session ranges between a few minutes and a few hours. If transmission is continuous, the potential savings in transmitted volume are significant.
- Inter-domain transmission lacks a direct relationship between content provider and access network domains on which a consensus for technology choice can be founded.
- Users cannot be expected to interact with non-trivial network functions, such as group management functions, unless explicit support by their local administration (i.e., the network operator) is provided. Likewise, the introduction of special software packages to support multicast on the users' side acts as a deterrent from using the service.
- The awareness to privacy issues with services provided in the Internet needs to be addressed such that the members of a communication group (e.g., a group simply receiving content in a 1:n communication pattern) may retain anonymity within the group.

Our construction of the MEADcast protocol, described in Section IV, aims to exploit these opportunities and to address the challenges in the scenario.

III. Related work

The idea of multicast was introduced decades ago and has drawn research efforts broadly. A variety of solutions have been proposed and a selection is presented here.

Traditional group-managed multicast [2], [3], [4], [12] specifies the transmission of an IP datagram to a *host group*, a set of zero or more hosts identified by a single IP destination address. It requires network support (multicast-capable routers) and the receivers to proactively join the host group. The routers and end-points use the Internet Group Management Protocol (for IPv4) or Multicast Listener Discovery (for IPv6) to maintain the multicast group. The deployment of IP multicast in the Internet is still far behind expectations due to a number of long-standing issues [13]. Amongst those are the management complexity put on the end-point and the requirement of global updates to routers.

An interesting approach on routing multicast traffic through a "multicast domain", namely Bit Index Explicit Replication (BIER) is presented in [7]. A multicast packet entering the domain is encapsulated in a BIER header by the ingress router, who then determines the set of egress routers to send the packets to them. Egress routers are represented by a bitstring in the BIER header. BIER simplifies the traditional multicast by eliminating the per-flow state and the explicit tree-building protocols with the trade-off of additional management layers introduced in BIER-compliant routers including the routing underlay, the BIER layer and the multicast flow overlay. A router has to maintain the routing information with other routers in a BIER domain besides the ordinary unicast one and be able to determine a set of egress routers for an incoming multicast packet, this indicates an extra state to be maintained. In comparison, a MEADcast router is much simpler by virtue of being totally stateless. Moreover, BIER limits its scope in routing multicast traffic within a BIER domain, leaving the inherent management issues of multicast untouched. In contrast, MEADcast concentrates all management initiative with the sender, allowing middle-boxes, non-MEADcast routers and receivers to remain agnostic of the use of multicast.

Common practice to overcome the traditional multicast management burden but still achieve better performance than ordinary unicast is to employ Application Layer Multicast (ALM), which implements multicasting functionality at the application layer instead of at the network layer by using the unicasting capability of the network. In contrast to the slow deployment of IP multicast, ALM gains practical success thanks to the ease of deployment. A survey of ALM over the period 1995-2005 was given in [14]. ALM's common approach is to establish an overlay topology of unicast links between the multicast participants where multicast trees can be constructed. The drawback of ALM is that the privacy of receivers is not ensured, which means the identity of one end-point might become known to the other; furthermore, the data delivery of ALM depends on the end-point capability, which could not guarantee the stability and reliability. MEADcast overcomes the above issues by leaving receivers agnostic of the underlying technology being used. ALM also requires, by principle, the deployment of ALM-capable software on at least a subset of the hosts participating in a multicast group.

Xcast [10] is a multicast scheme with explicit encoding of the destinations list in the data packets, instead of using a multicast group address. Xcast is able to support a very large number of small multicast sessions, making up the complementary scaling property to traditional IP multicast, as the latter has a scalability issue for a very large number of distinct multicast groups. Xcast transmits data along optimal route without traffic redundancy in the sufficient presence of Xcast-capable routers; otherwise, some special mechanisms have to be employed, e.g., tunneling or end-point upgrade, which introduce new management tasks for Xcast routers (exchanging and maintaining Xcast routing information) or end-points (performing network functions of an Xcast-router). In case an end-point assumes the Xcast router's functions due to the lack of network support, the identities of all receivers in a session are exposed to this one, raising the privacy issue and possibly hindering Xcast adoption by some applications. By design, an Xcast-router suffers from complex header processing: it has to perform a routing table lookup for each destination in the Xcast header's address list. Besides, Xcast is intended for multicast sessions of small number of participants (i.e., small group size), confining its suitable applications. Xcast6 Treemap [15], [16], a derivative of Xcast, while providing some enhancements to improve the traffic transmission latency in a use case of Xcast gradual deployment, still shares the same aforementioned limitations. However, Xcast does introduce many efficient features: no maintenance of multicast state by routers, routing based on ordinary unicast and automatic reaction to unicast reroutes, easy security and accounting, flexibility, among others. MEADcast takes advantage of some valuable concepts from Xcast while off-loading the management burden to the sender and simplifying the router functions.

IV. Protocol design

MEADcast is implemented by senders and routers. We describe the functions relevant for sender and router elements and message types and procedures necessary for the realization of these functions. A simple multicast scenario described in full illustrates the behaviour of the protocol.

The information needed to describe the protocol is

- the sender S,
- the set of end-points E_i to which S transmits data,
- the set of MEADcast routers R_j in the network,
- the MEADcast distance d (or MEADcast hop-count) from a MEADcast router to the sender in hops between MEADcast routers.

Association is indicated by superscript, i.e., a router responsible for a sub-set E_k of the end-points is R^k and an end-point served by a router R_j is E^j.

A. Functions

In MEADcast, we need to distinguish two groups of functions for the sender and the router.

Sender functions include transmission of *unicast* and *multicast* messages, *initiation of discovery* of MEADcast routers on paths to end-points and *discovery response evaluation*.

Router functions include normal *forwarding*, *decomposition* of multicast packets to unicast packets and multicast packets and *reaction to discovery requests* from a sender.

1) Discovery-related functions: Both the sender and the MEADcast router are involved in the discovery process. The goal of discovery is for the sender to determine the sequence of routers $(R^i_1, R^i_2, \cdots)$ on the path to each end-point E_i. Discovery requests and responses can be written as *req(E, d)* and *resp(E, d, R)*, respectively.

To initiate discovery, the sender addresses a MEADcast discovery request *req(E, 0)* to an end-point E. When receiving the request, every router R on the path to E increments d and forwards the discovery request *req(E, d+1)* to the next hop; at the same time, R sends a discovery response *resp(E, d+1, R)* to S. Thus, the first router R_1 on the path to E will send $(E, 1, R_1)$, the second $(E, 2, R_2)$ and so on.

S can compile the sequence $\{(E_i, d_1, R^i_1, d_2, R^i_2, \cdots), \cdots\}$ and can compute the group of end-points to be handled by a given router with a specific distance $(R_j, d_j, E^j_1, E^j_2, \cdots)$.

2) Decomposition: Decomposition, which is specific to MEADcast router, means the transformation of a multicast packet addressed to a set of target end-points into multiple unicast and multicast packets with the same payload.

The target addresses $R_j, E^j_1, E^j_2, \cdots, R_k, E^k_1, E^k_2, \cdots$ within a multicast message are structured to denote that a router R_i is responsible for end-points E^i. During decomposition a router will send unicast packets to each of

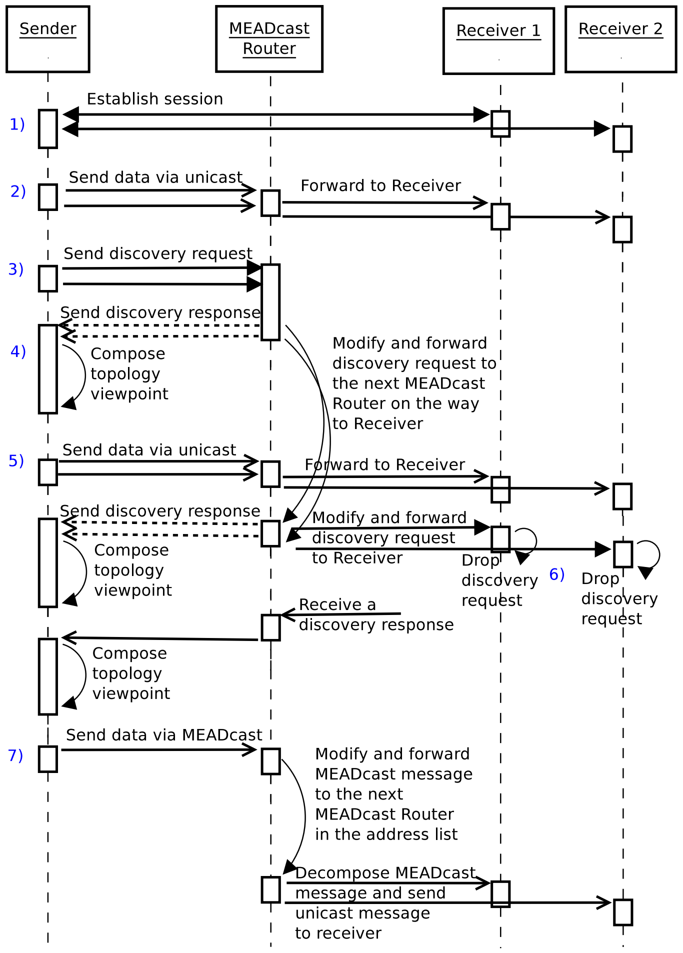

Sender
MEADcast Router
Receiver 1
Receiver 2
1)
Establish session
2)
Send data via unicast
Forward to Receiver
3)
Send discovery request
Send discovery response
4)
Compose topology viewpoint
Modify and forward discovery request to the next MEADcast Router on the way to Receiver
5)
Send data via unicast
Forward to Receiver
Send discovery response
Compose topology viewpoint
Modify and forward discovery request to Receiver
Drop discovery request
6)
Drop discovery request
Receive a discovery response
Compose topology viewpoint
Send data via MEADcast
7)
Modify and forward MEADcast message to the next MEADcast Router in the address list
Decompose MEADcast message and send unicast message to receiver

Figure 5. Interaction between MEADcast's entities.

the end-points it is responsible for and send multicast packets to the routers responsible for the remaining target end-points.

When a multicast packet is created, the targets already served either by unicast or by other multicast messages are removed from the list of targets of the packet being created. The removal process can be implemented efficiently by marking removal in a bitmap and thus eliminating the need to compose a new list of targets.

B. Interaction between MEADcast's entities

Figure 5 shows the interaction between the sender, the MEADcast router and the receivers, which involves the following steps.

1) First, the session is established between the sender and the receivers. In essence, this could be the request on data sent from each receiver to the sender or a data push from sender to some pre-defined targets. The double-headed arrow is used in this step to indicate both possibilities of a session initiator (sender or receivers).
2) The sender starts transmitting data to the receivers via unicast. Each router on the path takes part in the data transmission process as usual.
3) In the meantime, the sender also performs the MEADcast discovery by sending *MEADcast discovery request* to each receiver. Upon receiving a request message, the MEADcast router sends a *MEADcast discovery response* back to the sender and a *MEADcast discovery request* towards the receiver. The *MEADcast hop-count* in each message is modified indicating the *MEADcast distance* from the current router to the sender.
4) On receiving a *MEADcast discovery response*, the sender will compose its topology view, which is an overlay network connecting the sender, the MEADcast routers and the receivers.
5) The data transmission by unicast is still carried out during the MEADcast discovery phase.
6) The receivers drop *MEADcast discovery request* messages since they do not understand them.
7) After a pre-defined timeout, the sender stops transmitting data via unicast and starts the *MEADcast data sending* phase. MEADcast data messages are built based on the current topology viewpoint of the sender and are then transmitted. Each MEADcast router on the path processes the message according to the encoded MEADcast information. It may forward the message intact or decompose the message into multiple ones and forward them to the other MEADcast routers, or build and send unicast messages to the receivers.

C. Sender behaviour

The sender behaviour involves two phases: *MEADcast discovery* and *MEADcast data sending*.

The sender sends *MEADcast discovery requests req*$(E_i, 0)$ to all receivers and updates the network topology in the form of $(R_j, d_j, E_1^j, E_2^j, \cdots)$ whenever it receives a *MEADcast discovery response*. In the mean time, the sender also transmits unicast data to these end-points.

The *MEADcast data sending* phase starts when the discovery phase is complete (i.e., after a pre-defined timeout). Based on its network topology view, the sender constructs and transmits *MEADcast data messages* containing the target addresses $(R_j, E_1^j, E_2^j, \cdots, R_k, E_1^k, E_2^k, \cdots)$ for those receivers that can be served by MEADcast routers and stops unicast data to them. If the set of receivers of a multicast is larger than the maximum number of addresses encodable in the MEADcast header, the sender will divide the receivers into suitable subgroups, each served by its own MEADcast packet. If discovery reveals that a given MEADcast router would only be responsible for a single receiver, that receiver is served unicast in order to conserve header space in MEADcast transmission.

It is obvious that if there is no MEADcast router responsible for any receivers, the data sending phase of MEADcast operates exactly as unicast.

The discovery phase is carried out periodically so that the sender can maintain an updated view of the topology.

D. Router behaviour

A MEADcast router is an IP router with added control plane behaviour. It is capable to fulfil two tasks: participate in the discovery mechanism and process MEADcast packets that transport payload. The router does not hold state with respect to the topology, the presence or the location of other routers supporting MEADcast; nor does it hold information about the multicast groups.

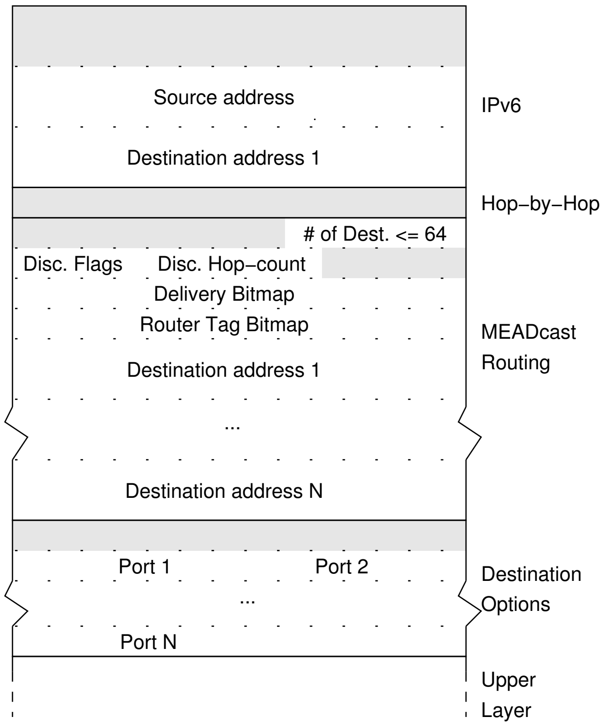
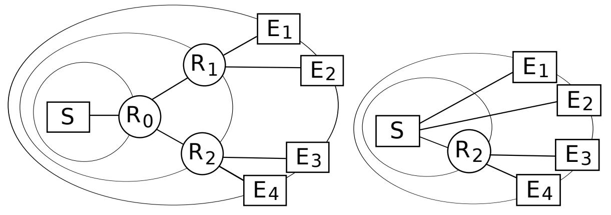


Figure 6. MEADcast header sequence with relevant fields.

1) The discovery mechanism: Discovery requests pertain to a path to one receiver. They are therefore addressed to that receiver and have the source address of the sender performing discovery. Upon receiving such a request, the router increments the MEADcast hop-count field of the request packet before processing (and forwarding) it as any other IP packet. In addition, the router transmits a discovery response to the sender, transmitting its own address, the hop-count of the discovery request and the address of the receiver that the discovery is for. In consequence, the sender is capable of discerning the sequence of MEADcast-capable routers on the path to a given receiver, as it is able to order the routers by the MEADcast hop-count of their responses.

2) The processing of multicast packets: relies on the address information transported in the MEADcast header. This information consists of a list containing both the addresses of receivers and those of the MEADcast routers expected to process the packet. A static *router tag bitmap* within the header differentiates between the addresses belonging to receivers and those belonging to routers. An additional bitmap, the *delivery bitmap*, is employed to mark the addresses already accounted for within the multicast tree.

On receiving a MEADcast packet, the router determines with the help of the router bitmap the addresses that it is responsible for. For the addresses of receivers (e.g., those in directly connected networks) it creates unicast packets and introduces them into its output queues. For the addresses of routers that are expected to perform additional distribution, it duplicates the received packet. In the duplicates, it marks all the addresses that it has accounted for in the delivery bitmap. Thus, the next router in the chain can determine that it should neither send unicast to the receivers at those destinations nor create multicast packets for the routers among them.



(a) Full network support. (b) Partial network support.

Figure 7. Network topology from sender viewpoint.

E. Protocol headers

The tasks of sender and router are supported by a number of data structures beyond those of the IP header. MEADcast for IPv6 uses the extension header mechanism to introduce multicast information. Figure 6 shows an overview of the sequence of headers employed in a MEADcast transmission with standard fields removed for clarity. The whole header includes the standard IPv6 header, Hop-by-Hop Options header, MEADcast Routing header and Destination Options header.

The fields depicted in the figure have the following purpose and structure:

- Source address and Destination address $1 \cdots n$ are normal IPv6 addresses.
- $\#$ of Dest: Number of destinations in IPv6 address encoded in the MEADcast Routing header.
- Disc. Flags: Discovery flags to mark a MEADcast message as discovery request, response or data delivery.
- Disc. Hop-count: Discovery hop-count, which is the MEADcast distance from a MEADcast router to the sender.
- Delivery Bitmap: to mark if a MEADcast router has received the MEADcast message.
- Router Tag Bitmap: to mark the position of the MEADcast routers in the MEADcast Routing header's address list.
- Port $1 \cdots n$: transport protocol port number (specifically UDP port number) for receivers.

F. Protocol mechanics by example

Figure 7 describes the network where the proposed scheme is effective, consisting of five end-points S, E_1, E_2, E_3, E_4 and three routers R_0, R_1, R_2. Their connections are shown in Figure 7(a) (without the rings). Two scenarios are described, the first one with all routers being MEADcast-capable, the second one with only R_2 being a MEADcast router.

1) All routers are MEADcast-capable: The communications between the sender S and the receivers E_1, E_2, E_3 and E_4 via the network in the first scenario are as follows.

1) S transmits unicast data to E_1, E_2, E_3, E_4.
2) S sends four different *MEADcast discovery requests req(E_i, 0), $i \in \{1, 2, 3, 4\}$.*
3) For unicast messages, R_0, R_1, R_2 simply forward it to the intended receiver.
4) R_0 receives *req(E_1, 0)*, it reacts to the presence of the Hop-by-Hop header and analyses the content of the MEADcast header. R_0 sends a *MEADcast discovery response resp(E_1, 1, R_0)* to S. It also sends *req(E_1, 1)*

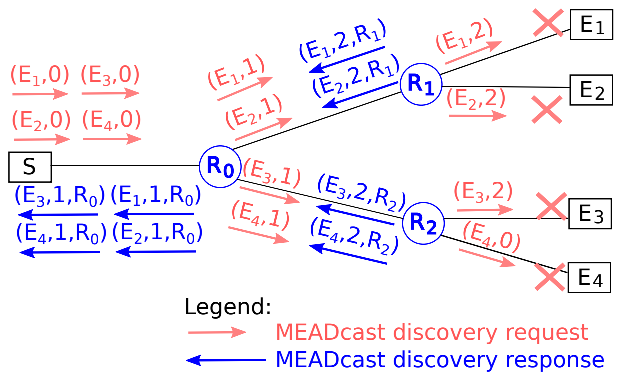


Figure 8. Discovery in all MEADcast network (case a).

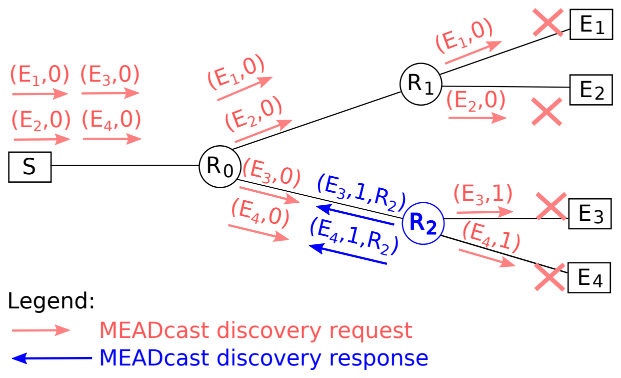


Figure 10. Discovery in sparse MEADcast network (case b).

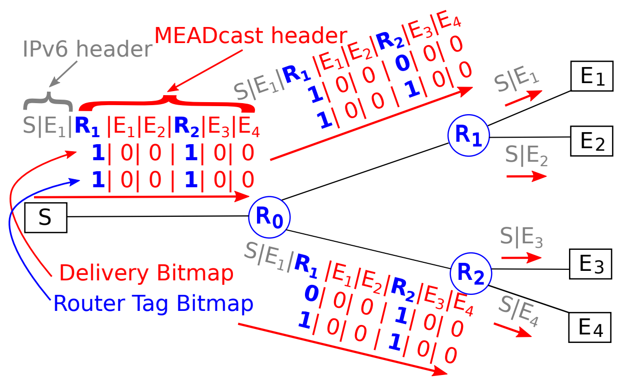


Figure 9. Data delivery in all MEADcast network (case a).

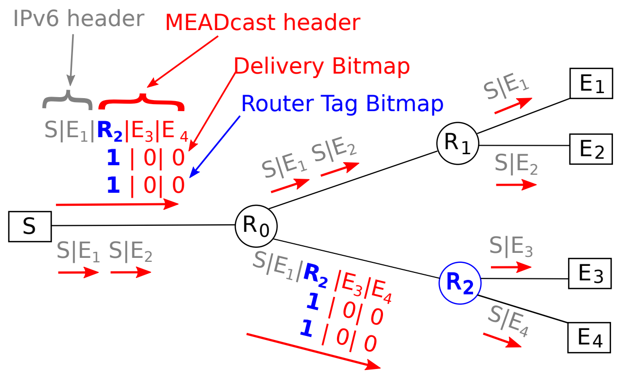


Figure 11. Data delivery in sparse MEADcast network (case b).

to E_1. The same procedure is carried out for *req*($E_2, 0$), *req*($E_3, 0$), *req*($E_4, 0$).

5) R_1 receives *req*($E_1, 1$), it sends *resp*(E_1, 2, R_1) to S. It also sends *req*($E_1, 2$) to E_1. The same procedure is carried out when R_1 receives *req*($E_2, 1$).
6) R_2 receives *req*($E_3, 1$) and *req*($E_4, 1$), it sends *resp*(E_3, 2, R_2) and *resp*(E_4, 2, R_2) to S. It also sends *req*($E_3, 2$) to E_3 and *req*($E_4, 2$) to E_4.
7) E_1, E_2, E_3, E_4 receive the unicast messages normally. For the *MEADcast discovery request*, they do not understand and simply drop it.
8) S receives *MEADcast discovery responses* and updates its network topology viewpoint as $(R_0, 1, E_1, E_2, E_3, E_4), (R_1, 2, E_1, E_2), (R_2, 2, E_3, E_4)$. The topology viewpoint of sender can be illustrated by the rings in Figure 7(a), where the sender is at the center, R_0 lies on the first ring with a distance of one, R_1 and R_2 are on the second ring with a distance of two and all receivers are always at the outermost ring.

All the steps above are depicted in Figure 8. The corresponding data delivery phase described in the next steps is shown in Figure 9. MEADcast routers and their associated entries in bitmap fields of MEADcast headers are marked in blue bold text. The same marking scheme is used in Figures 10 and 11.

9) S stops transmitting data via unicast and starts MEADcast data sending phase. S transmits a *MEADcast data message* consisting of:
   - E_1 as the destination IP address,
   - $\{R_1, E_1, E_2, R_2, E_3, E_4\}$ in the *MEADcast Routing header*'s address list,
   - *Router Tag Bitmap* being 100100, where bit 1 indicates a router and 0 an end-point, *Delivery Bitmap* is initialized with the same value of *Router Tag Bitmap*. Note that, *Router Tag Bitmap* also points out, which MEADcast router is responsible for which end-point, in this case: R_1 is responsible for E_1, E_2 and R_2 for E_3, E_4. *Router Tag Bitmap* of the *MEADcast data message* stays unchanged after leaving the sender.
10) R_0 receives the *MEADcast data message*, sees that:
   - it does not have to deliver message to any receivers since its address is not in the MEADcast address list,
   - based on the *Router Tag Bitmap* and *Delivery Bitmap* fields, there are two other MEADcast routers, namely R_1, R_2, needing to receive *MEADcast data message*. R_0 duplicates the original *MEADcast data message*.
     - The *Delivery Bitmap* field of the first one is modified to be 100000, indicating that R_2 has received a *MEADcast data message*. R_0 sends this message to R_1.
     - R_0 changes the destination IP address of the second message to E_3, modifies the *Delivery Bitmap* field to be 000100, indicating that R_1 has received a *MEADcast data message* and sends it to R_2.
     - The checksum at the transport layer (specifically, UDP) of each message is changed accordingly and is discussed further in Section VI-B.
     - The *Router Tag Bitmap* fields in both messages are the same as in original one.
11) R_1 receives a *MEADcast data message*, reads the *Router Tag Bitmap* field and sees that it is responsible for E_1 and E_2. R_1 constructs two unicast messages with the

data from the *MEADcast data message* and transmits each to E_1 and E_2. The checksum at the transport layer of each message is changed accordingly. The *Delivery Bitmap* value of 100000 shows that there is no other MEADcast router who needs to receives this *MEADcast data message*.

12) R_2 receives a *MEADcast data message*, reads the *Router Tag Bitmap* field and sees that it is responsible for E_3 and E_4. R_2 constructs two unicast messages with the data from the *MEADcast data message* and transmits each to E_3 and E_4. The checksum at the transport layer of each message is changed accordingly. The *Delivery Bitmap* value of 000100 shows that there is no other MEADcast router who needs to receives this *MEADcast data message*.

2) Only R_2 is MEADcast-capable: The communications between the sender S and the receivers E_1, E_2, E_3 and E_4 via the network in the second scenario (only R_2 is a MEADcast router) are sketched in Figures 10 and 11, which have the same first three steps as in the first scenario. The further steps are as follows.

1) R_0 receives *req*($E_1, 0$), it reacts to the presence of the Hop-by-Hop header and analyses the content of the MEADcast header, which it does not understand. It forwards the message further to the E_1 direction. R_0 does not drop the message since the option type identifier of MEADcast header is 00 [17]. The same procedure is performed for *req*($E_2, 0$), *req*($E_3, 0$), *req*($E_4, 0$).
2) Similarly, R_1 receives *req*($E_1, 0$) and *req*($E_2, 0$), it sends these messages to E_1 and E_2.
3) R_2 receives *req*($E_3, 0$), *req*($E_4, 0$). It sends *resp*(E_3, *1*, R_2) and *resp*(E_4, *1*, R_2) to S. It also sends *req*($E_3, 1$) to E_3 and *req*($E_4, 1$) to E_4.
4) E_1, E_2, E_3, E_4 receive the unicast messages normally. For the *MEADcast discovery requests*, they do not understand and simply drop them.
5) S receives *MEADcast discovery responses*, updates its network topology viewpoint as $(R_2, 1, E_3, E_4)$. Its network topology viewpoint is illustrated in Figure 7(b). There is no MEADcast router on the paths to E_1, E_2, only R_2 lying on the first ring of distance one is on the paths to E_3, E_4. All receivers are on the outermost ring.
6) S starts MEADcast data sending phase. Based on its topology view, S sees that:
   - there is no MEADcast router on the paths to E_1, E_2. S unicasts data to these receivers.
   - R_2 is on the paths to E_3, E_4. S transmits a *MEADcast data message* with E_3 as the destination IP address and $\{R_2, E_3, E_4\}$ in the *MEADcast routing* header's address list, *Router Tag Bitmap* and *Delivery Bitmap* being 100.
7) For unicast messages, R_0, R_1 simply forward them to the intended receiver.
8) R_0 receives the *MEADcast data message*, which it does not understand. It forwards the message further to the E_3 direction. R_0 does not drop the message since the option type identifier of MEADcast header is 00 [17].
9) R_2 receives a *MEADcast data message*, reads the *Router Tag Bitmap* field and sees that it is responsible for E_3 and E_4. R_2 constructs two unicast messages with the data from the *MEADcast data message* and transmits each to E_3 and E_4. The checksum at the transport layer of each message is changed accordingly.

G. Sender API

All state information about the multicast tree is held by the sender, as MEADcast receivers are multicast-agnostic by design. Routers with MEADcast-capability need not keep state about the multicasted flows but require only the addition of handling the discovery mechanism of MEADcast and the processing of MEADcast data packets.

The sender has special requirements on the API by which the network socket is controlled. We build on the work described in RFC 7046 [18] ("Hybrid Adaptive Multicast", HAM; "Transparent Hybrid Multicast") as a generic API that aims to provide a common vertical interface for multicast sockets without regard to the multicast technology. Although the HAM API does not specifically target the sender-centric multicast schemes and does require extensions to fully support them, the expressive power of the API is sufficient to control most aspects of the sender and does not contain aspects that are unimplementable for MEADcast.

The HAM model of multicast is that an application uses a HAM socket, which can be bound to network interfaces and multicast groups to support an n:m communication pattern, as known from "traditional" multicast. The application effects the life-cycle management of the socket, its binding to network interfaces and the joining and leaving of multicast groups. Multicast group identifiers are separated from network addresses and noted as a URI (Uniform Resource Identifier) in order to avoid assumptions about the actual multicast technology being employed. As a tacit assumption, every host supporting the API is expected to manage its membership in a group: it is this tacit assumption that will require extension of the HAM API in order to support MEADcast.

We iterate through the four interface sections of the HAM API and apply them to MEADcast, before specifying the necessary extensions.

1) Group management: The `create()` and `delete()` calls act on abstract Multicast Sockets as described in the RFC.

The group management functions `join()` and `leave()` are strictly speaking not required for MEADcast, since group membership is only managed by the sender. They are restricted to enabling MEADcast to change a group's denomination during a session.

The calls for (de-)registering a multicast data source, `srcRegister()`, `srcDeregister()` duplicate the function of the socket creation and deletion functions for the purposes of the 1:n MEADcast.

2) Send and receive: The `send()` function signature can be applied as is. The `receive()` function is inert, given that the sender does not receive multicast and the receivers do not support the API.

3) Socket options: RFC 7046 specifies a number of socket management functions, which are useful for a sender application and can be supported in a MEADcast sender. The functions allow the management of interfaces bound to a socket with `getInterfaces(), addInterface()` and `delInterface()` and setting of network parameters for sockets/interfaces with `setTTL(), getTTL`.

The function `getAtomicMsgSize()` *"returns the maximum message size that an application is allowed to transmit per socket at once without fragmentation"*. Given that MEADcast performs a trade-off between this "atomic size" and the number of multicast targets addressable with a single PDU (Protocol Data Unit), applications should always consult this function to determine the application SDU (Service Data Unit) size offered by the MEADcast trade-off.

If an application cannot practically reduce the amount of data sent at once, it is useful to allow it to influence the trade-off performed by the MEADcast socket, i.e., to sacrifice the number of addressable targets in favour of larger messages, e.g., in order to fit frames of media streams into one message. We present an extension to support such hints later in Section IV-G5.

4) Service Calls: The service calls of the HAM API allow an application to retrieve information about the multicast environment of a HAM node.

The most relevant is the function `childrenSet()`, which returns the clients (receivers) served by this sender, queried by interface. For a complete list of receivers, the list of interfaces retrieved with `getInterfaces()` can be iterated through calls of `childrenSet()`.

The maximum message size transmittable (with possible fragmentation) over a socket can be queried with `getMaxMsgSize()`. For MEADcast, the value returned can be the maximum size transmittable by unicasted IP; an application sending messages of that size would either force transmission by unicast or the multicast of fragments.

Some service calls return information possibly useful to the application: `groupSet()` would retrieve the groups mapped to a given interface; the application can request the neighbour nodes (assumed to be those on the same link layer segment) by means of calling `neighborSet()` for an interface.

Some of the service calls lack a function for MEADcast but can be implemented to return sensible values for the application. The `designatedHost()` function determines *"whether this host has the role of a designated forwarder (or querier), or not."*. The `parentSet()` function returns a list of neighbours, from the node (in our case, the sender) receiving multicast; a MEADcast implementation can return an empty set.

Membership events are issued from a HAM socket instance *to* the application to notify of *other* members of the multicast group joining or leaving the group. In our case, that information originates with the application. Hence, supplying it again can be useful only for confirmation of changes to the group made within the MEADcast layer. Event flow management with `enableEvents()` and `disableEvents` can be implemented as specified.

5) Extension: The HAM API document does mention, in short, Xcast as an example for a sender-centric, agnostic-receiver-multicast protocol, and it sketches a few ideas for necessary information management for this class of protocols. The API itself, however, fails to take into account that Xcast as well as MEADcast require a means to inform a sender about the introduction or removal of a receiver to/from a group. Consequently, it is necessary to supply two fundamental functions in addition to those specified in RFC 7046.

a) Group management API extension: For the extension, we employ the same terminology and language as RFC 7046. In particular, the receivers of packets multicasted from one sender are termed *"children"* with respect to that sender.

The application can register or deregister a child at a socket. These functions are mandatory, as they constitute the only manner in which MEADcast can be made aware of status changes in children, i.e., in receivers.

```
joinChild (in Uri groupName,
           in Child childSpec,
           out Int error);
```

and

```
leaveChild (in Uri groupName,
            in Child childSpec,
            out Int error);
```

In each function, `groupName` identifies the multicast group that the receiver ("child") is to be added to. The implementation is responsible for adding it to the appropriate socket or sockets and, if enabled, to generate Events to this effect. An `error` return value of 0 indicates success, one of -1 indicates failure.

The `Child` structure contains a mandatory set of information items pertaining to the basic multicast service and a set of optional items to support optimisation or application-specific parameters. It is given as:

```
Child { Int IPversion ;
        IPAddress childIPAddress ;
        IPAddress routerIPAddress ;
        Int port ;
        Int pathMTU ; }
```

The `IPversion` mandatory field identifies the Internet Protocol version to be used when casting to this receiver. The `IPAddress` type is a binary encoded IPv6 or IPv4 (according to the value of `IPversion`) unicast address. The mandatory field `childIPAddress` identifies the receiver by IP address.

The optional field `routerIPAddress` identifies the initial router (if any) to be responsible for transmitting unicast to the receiver, giving its IP address on the path between sender and receiver. The optional field `pathMTU` contains the maximum transfer unit (MTU) on that same path. The optional field `port` carries the transport layer protocol port at the receiver.

The structure is intended to be extensible with respect to the optional items. Conceivable options include the receiver's preferences regarding quality, transmission volume, as well as accounting parameters.

b) Hints: It is necessary to control the trade-off between the number of targets addressable in a single MEADcast PDU and the maximum size of messages acceptable from upper layers and applications while avoiding fragmentation. While the MEADcast stack might strive to maximise the number of

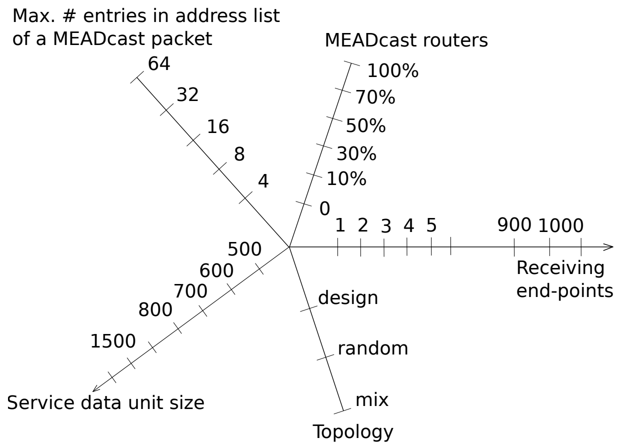


Figure 12. Parameters for experiments.

addressable targets, application-specific information regarding the structure of data can be helpful. For that purpose, we extend the API with an additional, optional function:

```
setExpectedSDUSize (in Int sduSize,
                    in Uri groupName,
                    out Int error);
```

The parameter `sduSize` is the size of the service data unit (SDU) of the upper layer, in bytes, that the application expects to `send()` to a group specified by `groupName`. The function returns a value of zero, if the size can be transmitted without fragmentation or -1 if the size cannot be transmitted.

Note that the SDU will encompass the payload and the transport protocol header: it is up to the application programmer, who decides on the transport protocol to be used, to calculate the value.

For example, an application transmitting a media stream can set the expected size of the payload to be the size of one or more media frames plus the size of an UDP header. In response, the MEADcast implementation can determine the maximum number of addresses included in its own header.

V. Evaluation

We have performed experiments within the parameter space illustrated in Figure 12. We simulate MEADcast for 100 routers both on random network topologies with a diameter of 16 (generated by GT-ITM [19]) and on topologies designed according to realistic scenarios using ns-2 [1]. To be more specific, the experiments are carried out by the following steps.

1) A core network of 100 routers is generated.
2) Three core networks are created by randomly enabling the MEADcast capability on 30, 50, 70 routers of the core network in step 1, respectively. One more designed core network by selectively enabling the MEADcast capability on 10 routers and another for all routers are added. In total, there are six core networks associated with 0, 10, 30, 50, 70 and 100 MEADcast-capable routers, in which the second one is intended for designed cases and the others for random ones.
3) The end-points are added to the core networks in step 2. The number of end-points ranges in 100, 200,...,1000. We actively distribute the end-points over MEADcast-capable routers for design cases while they are scattered arbitrarily in random cases. Each association of a chosen end-point set with a core network in either case forms a final topology, which means there are 50 random final topologies and 10 designed ones.
4) For each final topology in step 3, two experiments, one for unicast and another for MEADcast, are performed. An end-point is chosen as sender and the remaining end-points are receivers. The sender then transmits a data stream of 800 MB into the network to all the receivers. The trace file created by *ns-2* for each experiment to log the whole data transmission process is analysed. The traffic volume on all links of the whole network is calculated and logged for further analysis. It is 120 different experiments on the whole.

A screenshot of a random topology with 100 routers (red square nodes) and 100 end-points (grey round leaf nodes) is presented in Figure 13. Node 100 (leftmost leaf node) is chosen as sender, the remaining leaf nodes are receivers.

Table I shows an excerpt of the whole result when experimenting MEADcast and unicast for 100, 500 and 1000 end-points. Their corresponding total data volume in this table is plotted in Figure 14 whereas Figure 15 presents all the results. The percentage of traffic saving between MEADcast and unicast for the case of 1000 end-points is highlighted by double-headed arrows in the latter figure.

If there is no MEADcast router, the sender sends mainly unicast messages and periodically sends discovery messages that occupy only a little traffic volume over the whole network. This discovery overhead is indicated by the value "$-0.x$" in Table I and depends on how many times the discovery is performed. Hence, the traffic volume of the MEADcast protocol when there is no MEADcast router is approximately that of unicast, provided that sender has large traffic to send. The gap increases when the number of receivers and the percentage of MEADcast routers grow. The extreme case

Table I. Total traffic volume in the whole network [MB].
* indicates designed topology.

Topology					
End-points	*MEADcast routers*	*Unicast*	*MEADcast without discovery*	*Discovery (one time)*	*Traffic saving [%]*
100	0	761,504	761,504	0.075	0
	10(*)	761,504	271,104	0.149	64,4
	30	761,504	563,376	0.157	26
	50	761,504	433,504	0.210	43.1
	70	761,504	372,964	0.246	51
	100	761,504	272,420	0.500	64.2
500	0	4,136,544	4,136,544	0.409	$(-0.x)$
	10(*)	4,136,544	1,279,936	0.813	69.1
	30	4,136,544	2,513,296	0.897	39.2
	50	4,136,544	1,698,336	1.216	58.9
	70	4,136,544	1,069,276	1.389	74.2
	100	4,136,544	902,056	2.835	78.2
1000	0	8,341,776	8,341,776	0.826	$(-0.x)$
	10(*)	8,341,776	2,525,904	1.640	69.7
	30	8,341,776	4,978,384	1.802	40.3
	50	8,341,776	3,137,972	2.462	62.4
	70	8,341,776	1,866,456	2.819	77.6
	100	8,341,776	1,552,304	5.727	81.4

[1]https://www.isi.edu/nsnam/ns/

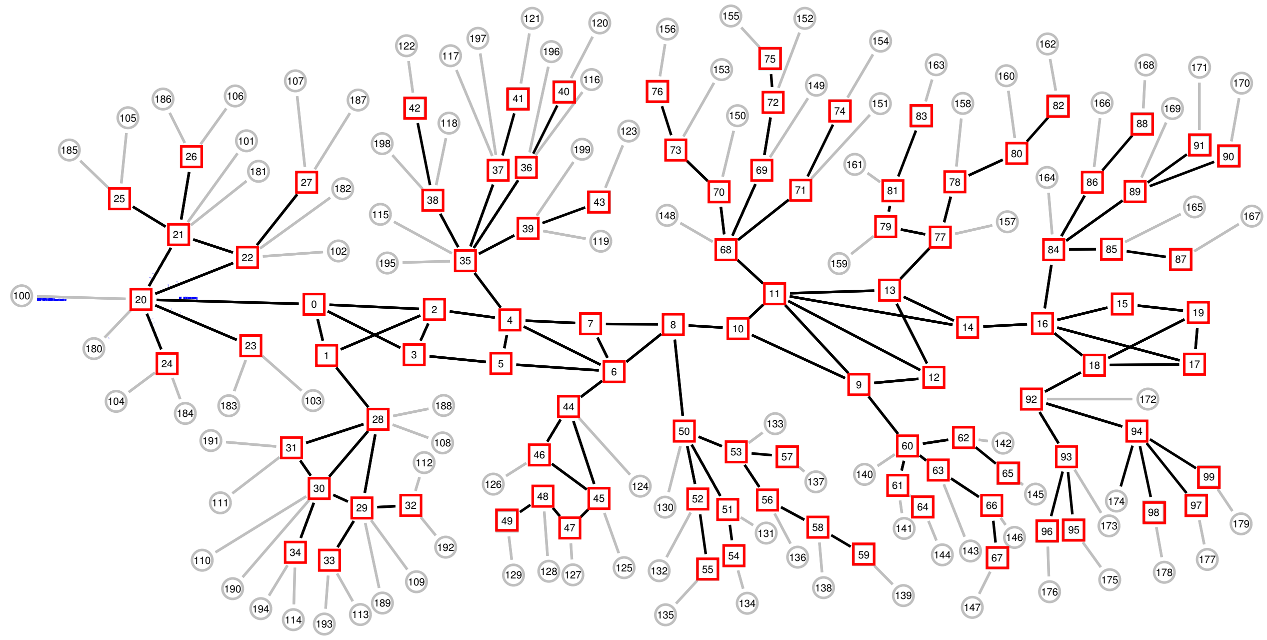


Figure 13. A network topology generated by GT-ITM.

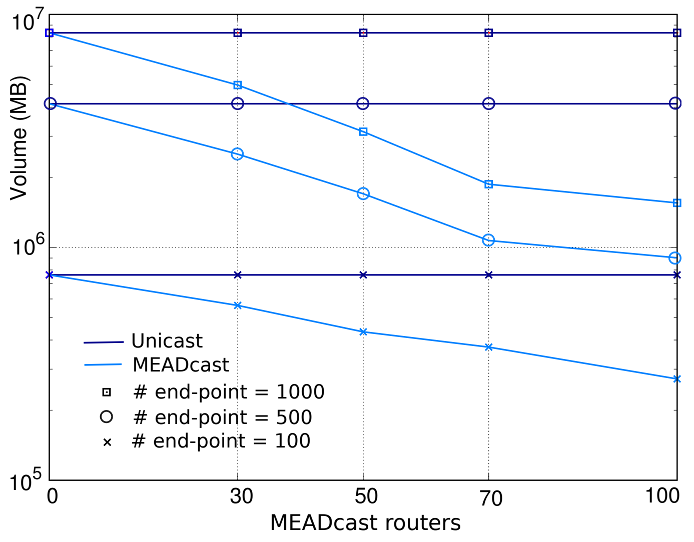


Figure 14. Total data volume of unicast and MEADcast - an excerpt.

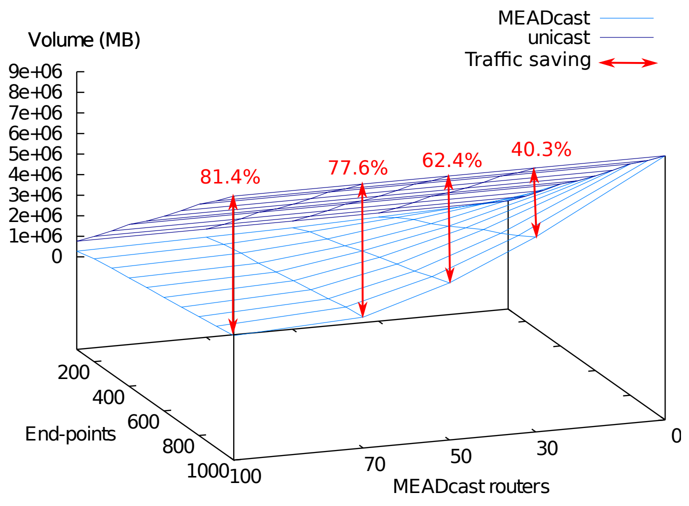


Figure 15. Total data volume of unicast and MEADcast when varying MEADcast support in the network and the number of receivers.

of 1000 end-points and 100% MEADcast routers shows the difference of 81.4% in total traffic volume.

The total traffic volume reduction is considerable in the presence of sufficient MEADcast routers, as shown by the designed cases. The link stress (i.e., the number of packets with the same payload sent by a protocol over each underlying link in the network) [20] at the sender is reduced to an even higher degree.

The impact of the service data unit size and the number of entries in the address list are discussed in Section VI.

VI. Discussion

We discuss a selection of aspects of MEADcast pertinent to its deployment, the relationship of the protocol implementation to other architectural components within the sender software stack, as well as limitations and the treatment of anomalies that may occur.

A. *Addressing scenario challenges*

The scenario (see Section II) describes challenges originating in organisational and technological distance between content providers in the sender role and the network operators hosting the receivers.

MEADcast allows the gradual shift from many unicast flows to a reduced number of multicast flows. Due to the protocol's properties, the receivers are not made aware of the use of the protocol or the manner in which it transports a flow to them. Certainly, the interception of discovery packets addressed to receivers allows them to actively detect the sender's ability to employ MEADcast but without discerning with certainty

- if multicast is actually in use for “their” flow, and
- which other addresses receive the same data stream.

The privacy of each receiver is thus preserved.

The content provider can deploy MEADcast within their own network in order to reduce the load between the senders and the MEADcast routers. The operators of neighboring networks to which the content provider organisation maintains contractual relationships can be informed of the option to deploy the protocol in order to improve distribution and thus reduce the transmission volume.

Our approach specifically targets the subspace of applications delivered via massive unicast into networks in different administrative domains than the sender. While MEADcast can be employed as a general-purpose multicast protocol, its benefits are most apparent in these scenarios: without an approach with native internetwork support and incremental deployment features it does not seem plausible that multicast would be deployed at all. In the special cases where a managed service is provided locally (e.g., in a symmetric conference-style application provided to participants within the same administrative domain) and pre-configured by that domain's network management, traditional, group-based multicast may be the more effective choice.

1) Data-driven deployment: MEADcast signalling, i.e., the discovery subprotocol, creates a technical means to advertise the technology being in use and at the same time to identify the number and location of receivers within a network to the network operator.

By observing MEADcast discovery traffic, a network operator is capable of determining

- the amount of traffic with potential MEADcast support
- whether the traffic terminates within the operator's network or constitutes transit traffic,
- the distribution of the terminating traffic within the network topology, and
- the distribution of the transit traffic onto egress points.

Based on this information, the network operator is capable to gauge the savings in terms of traffic volume if MEADcast capability is introduced at a certain point in the network.

Consequently, the mechanism allows MEADcast routers to be deployed at points where they have the greatest benefit, while maintaining the freedom from association between a content provider sending multicastable data and a network operator hosting multiple receivers.

2) Incentives for non-access networks:

a) Incentives for peer and transit networks: to support the protocol may originate in their own capacity management, in order to reduce volume destined to their own egress routers.

b) Carrier services: may not be motivated to support MEADcast, depending on their usage accounting model. However, the networks they serve might request MEADcast support as a service, when acting as transit networks to the MEADcast traffic or as the hosting network for many receivers.

3) Deploying m:n multicast applications: Applications like video conferencing (see Section II-C have an m:n (in their special case $m = n$) communication pattern. They can benefit from MEADcast by deploying the sender software stack on participants' hosts, thus placing those hosts in the role of a MEADcast sender.

This type of deployment has a number of beneficial properties:

- The sender stack is optional: if it is missing or inoperable on a given host, the outgoing traffic from that host is regular unicast.
- The support of the networks connecting the participants is employed to the degree that it is actually available, but network operators may decide to deploy MEADcast capability in order to manage the network load produced by video conference sessions.
- No mapping of participants' network identities to multicast groups is necessary, therefore allowing a conference to be configured by selecting, e.g., SIP identities.
- Finally, the extension software stack can be pre-configured by the administrative domain's system manager without regard to the location of other participants and without the need for user configuration.

B. Relation to upper layers

The concepts of MEADcast require a higher degree of interaction between the network layer and its upper layers (transport and application), which merits discussion. While our simulation results indicate significant performance gains for a wide range of parameters, MEADcast scenarios may be limited by properties of the protocol or the applications using it, and routers may experience a higher control plane load. After discussing these points, we conclude with remarks on fault and security issues.

Decomposition of MEADcast data packets may yield packets with different destination addresses and thus invalidate checksums in upper layer headers that include network addresses in the checksum (e.g., UDP for IPv6). For the new packet to be valid at the destination, MEADcast routers must re-compute these checksums for every new unicast packet and every new MEADcast packet with a different destination address. This issue is due to the re-use of network addresses in transport layer protocols, and problematic not only because of the increased load on routers' control plane but also because of the requirement to handle protocols other than IP. The checksum calculation in the transport layer by a MEADcast router is similar to that of an Xcast router, which is presented in [10]-Section 10.1.

Transport layer port numbers will differ at end-point sockets and have to be included in the MEADcast header along with the IP address of each end-point, thus creating an additional binding to the transport layer.

Network service primitives do not support addressing multiple receivers. Therefore, applications and higher protocols on the sender side must be modified to make use of MEADcast. A solution idea would be to use “regular” IGMP/MLD-based multicast on the first hop, thus allowing applications and higher protocols to employ multicast addressing as usual, then use a proxy function to translate between regular multicast and MEADcast before transmitting. While Path MTU discovery [21] is a standard function of the Internet, the application requirements on payload size are not readily available to allow the computation of optimum header size. We envision an interface to the network layer allowing the application to issue *hints* with respect to its intended use of the network.

We emphasize that these modifications are required for the sender only. The providers of asymmetric applications (IP-TV, Internet radio, etc.) can be assumed to correctly gauge the cost and benefit of introducing modifications to consolidate the multitude of unicast flows they create presently.

C. Limitations

Inherent limitations of the approach include the maximum number of entries in the address table, the overhead introduced by the address table, the time required to establish multicast structures and load introduced in the control plane of routers.

MEADcast routers do not keep group information, thus rendering MEADcast processing stateless, while nevertheless complex in contrast to multiple flows that may be handled by accelerators such as FPGAs.

MEADcast performs a gradual transition from a number of unicast packet flows to a (smaller) number of multicast flows as the availability of MEADcast-capable routers is discovered. Sessions that are shorter than the time for discovery not only forego the benefit of multicast but also carry the additional load for discovery; they are an application for unicast.

Given a path MTU value, the number of entries in the address table determines the remaining space for payload. If the service data units received from the upper layer is small, the sender may enlarge the number of entries, however, for large number of end-points even small payloads will require the sender to issue multiple multicast packets. Figure 16 shows the critical points where data volume is increased when the address table space of 32 entries is exhausted by one router multicasting to an increasing number of end-points. Conversely, a large address table leaves less space for payload and may lead to fragmentation, as illustrated in Figure 17.

D. Fault and security considerations

Packet loss naturally incurs a larger penalty for MEADcast than unicast, as more receivers are affected. In particular, the failure of a MEADcast path by changes in routing (by administrative action or by faults) will lead to continuous loss of packets until the periodic discovery mechanism informs the sender of the change in the network topology. A higher discovery frequency might lessen the consequences at the expense of increased control plane load in routers and an increase in the number of (albeit small) packets transmitted over a path.

Beyond the security issues noted for Xcast (see [10]), which also employs sender-based multicast, we note that the deprecation [22] of the Type 0 Routing Header in IPv6 to prevent amplification attacks suggests careful scrutiny of any mechanism that causes Internet routers to transmit more packets than they receive. We presume our mechanism to be reasonably safe due to the following properties: *i)* the total volume of transmitted multicast data does not exceed the corresponding unicast volume for the same data, with the exception of the lightweight signalling overhead required for consolidating multicast, *ii)* addresses are not modified by routers, i.e., data is transmitted via the same path in both unicast and multicast modes.

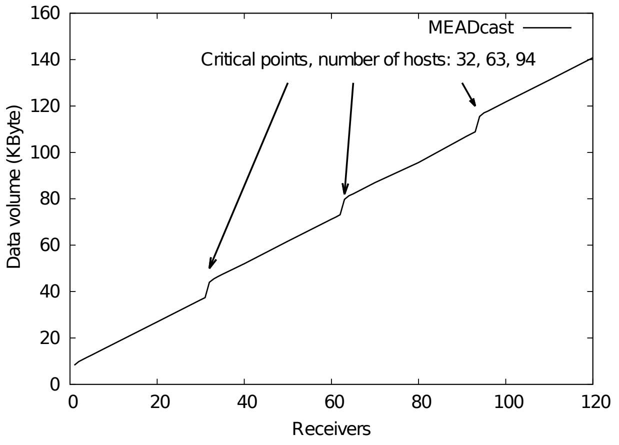


Figure 16. Total volume increased by exhausted address table.

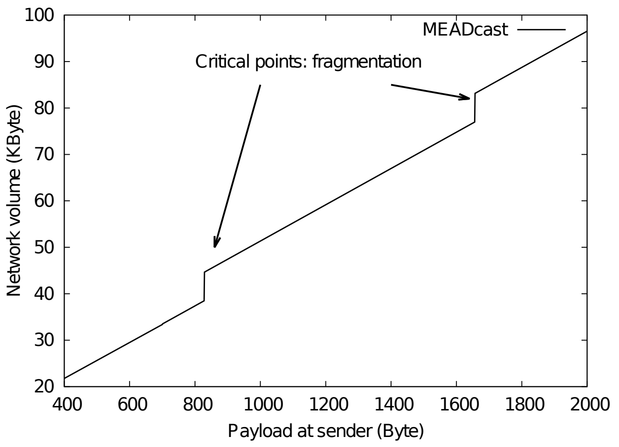


Figure 17. Fragmentation impact on total volume (1 router, 31 end-points).

RFC 6398 [23] warns that the Router Alert Option (RAO) [24], which is useful to indicate MEADcast packets to routers, may be used as an attack vector. The authors recommend, as a measure to defend against attacks, to either forward packets with RAO without evaluating the RAO content or, as a last resort, to drop packets with the RAO. Given that the RAO is employed by a number of well-known protocols, e.g., Resource Reservation Protocol (RSVP), Internet Group Management Protocol (IGMP), we assume that support for RAO will be maintained.

E. Multiple unicast paths between two MEADcast routers

As MEADcast routing relies on unicast, there are cases where different data delivery unicast paths from the sender to receivers cause a MEADcast router to send discovery responses to the sender with different distances (MEADcast hop-count). Figure 18 illustrates a case where the distance of R_3 to S is 4 according to the path from S to E_1 (S-R_1-R_2-R_4-R_3-E_1) while it is 3 for the path from S to E_2 (S-R_1-R_2-R_3-E_2). This case is conceivable in reality for some reason, e.g.,

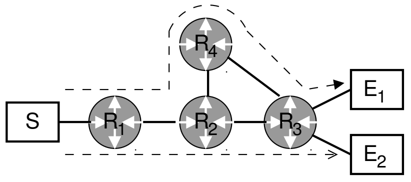


Figure 18. Hop-count conflict situation.

path load balancing deployment causes traffic to be directed on different paths. The network topology viewpoint building process at the sender is influenced: *i)* R_3 could have the same distance to S as R_4 and both are immediately after R_2 or *ii)* R_3 is one hop farther than R_4. Note that the sender's viewpoint is key for the MEADcast data sending phase and has to be a tree (with the sender as root), i.e., a loop-free viewpoint. Therefore, only one possibility of these two can appear in the final sender's viewpoint.

Several strategies could be employed by the sender in this case, e.g., for multiple discovery responses from a source (MEADcast router) with different distances (MEADcast hop-count), the sender could:

1) consider only the distance in the first response, which means this router always has this distance to the sender regardless of any different distance reflected in subsequent discovery responses,
2) choose the shortest distance extracted from all the discovery responses,
3) choose the longest distance,
4) choose the most popular distance, i.e., the distance with highest frequency in all responses.

Our current implementation is based on the second option since it intuitively induces the least latency in data delivery. The longer path can be seen as backup in case the shortest one is broken and comes into use after the next periodic discovery phase. So the path through router R_4 in Figure 18 is considered as redundant and the traffic is normally sent from S to E_1 and E_2 on the path R_1-R_2-R_3.

It is conceivable to amend the specification of router behaviour to effect the transmission of a responding router's IP addresses within the discovery response message, e.g., encoded in the destination address field.

F. Opportunities for optimisation

It is easy to upgrade MEADcast to control the data transmission's efficiency, e.g., which router is responsible for which receiver and how many of them. It is only the design matter at sender side, not at MEADcast router or other end-points. So the upgrade point is only the sender.

1) Reflection of unicast: It is safe to assume that the majority of routers in the Internet are either agnostic or phobic.

- If a datagram is forwarded through agnostic routers past the point where it should be branched, a receiving aware router can forward it back over the same path by unicast if it recognises this to be the case.
- The reflecting router should at the same time notify the sender of reflection being necessary on this path, e.g., by sending it an ICMP message containing the reflected destinations. In response to this message, the sender is expected to stop multicasting to those destinations and use unicast for them instead.
- The reflecting router should process (i.e., reflect) only a limited number of multicast datagrams from a certain source in order to avoid senders that are in violation of the protocol. Reflection, like multicast itself, might be a vector for an amplification attack, using a reflecting router to inject large amounts of traffic into the network.

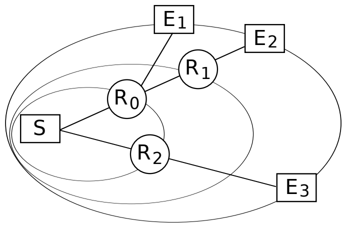


Figure 19. A scenario for optimization consideration.

2) Greedy unicast: If only few destinations are reachable via a given path, it may be more efficient to transmit unicast. E.g., it may not be worth the effort of transmitting and processing extra headers in the case where only one destination is addressed in the MEADcast header. Figure 19 raises such a scenario where according to the sender's viewpoint after the topology discovery process, there are some MEADcast routers (R_1, R_2) responsible for only one receiver. For the setting of allowing a MEADcast router to be responsible for at least one receiver, the data delivery tree is then exactly alike the topology viewpoint of the sender: the sender may build two different MEADcast messages for each branch: $\{R_0, E_1, R_1, E_2\}$ and $\{R_2, E_3\}$ and send them to the direction of E_2 and E_3, respectively. The data delivery process is similar to the steps described in Section IV-F. R_0 composes and sends a unicast message to E_1 and sends a MEADcast message to R_1, who then composes and sends a unicast message to E_2. Obviously, it is not so efficient compared to the case where the first message built by sender is: $\{R_0, E_1, E_2\}$, then R_0 will decompose the MEADcast message into two unicast ones and send them to E_1 and E_2. The transmitted message from sender to R_0 is smaller by saving up the space for R_1 in the address list, the unicast message sent from R_0 to R_1 is also more bandwidth-efficient compared to the MEADcast message in the former case. Similarly, S does not send MEADcast messages to E_3 but unicast ones, which is also more efficient.

VII. Conclusion

The MEADcast protocol introduced in this article addresses long-standing issues of the adoption of multicast. We find that the requirement on group participants to perform group management, as well as the addressing scheme and the intra-domain design of traditional multicast erect barriers to its adoption as a natural, common means for group communication. The massive unicast employed by many services, in consequence, leads to a significant waste of network resources due to the transmission of identical payload over all links to all receivers.

We have introduced MEADcast as a sender-centric multicast protocol that introduces a separation of the concerns of managing groups and holding state (at the sender), forwarding multicast payload without having to keep state (routers) and receiving the payload, without having to have knowledge of the network technology employed (receivers). The discovery mechanism of MEADcast allows a "fall-forward", incremental introduction of multicast transmission, depending on the actual support by the network, on a single receiver basis. At the same time, in contrast to a fall-back mechanism, it avoids the transmission of address tables to receivers and consequently avoids the potential breach of privacy between them. Our experiments have focused on the reduction of the total volume of traffic in the network compared with unicast in a wide range of simulated scenarios with varying support for MEADcast in the network. These experiments have employed both randomly generated topologies with randomized placement of MEADcast routers and topologies specified to reflect real-life networks. We have found that even a modest amount of MEADcast routers in the network yield a reduction in resource use. The gains in performance become larger with a higher degree of support.

Our discussion indicates several open questions and avenues for development, including the study of the load increase in router control planes and the real-world evaluation of streaming applications based on a module implementation for the Linux kernel and the development of an interface for the management of multicast groups and parameters on the sender side. We intend to employ the kernel implementation for studies of the protocol's behaviour during the experimental provisioning of services transported via MEADcast within a moderately controlled environment such as an academic network. A different point of interest is the realisation of MEADcast with virtual network functions, to be used in and between Software Defined Networks (SDN).

Acknowledgment

The authors wish to thank the members of the Munich Network Management (www.mnm-team.org), directed by Prof. Dr. Dieter Kranzlmüller, and Dr. Truong Khoa Phan for valuable comments on previous versions of this paper.

References

[1] C. N. Tran and V. Danciu, "Privacy-preserving multicast to explicit agnostic destinations," in *The Eighth International Conference on Advanced Communications and Computation (INFOCOMP 2018)*. IARIA XPS Press, 2018, pp. 60–65.

[2] S. Deering, "Host extensions for IP multicasting," RFC 1112 (INTERNET STANDARD), Internet Engineering Task Force, Aug. 1989, updated by RFC 2236. [Online]. Available: http://www.ietf.org/rfc/rfc1112.txt

[3] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, "Internet Group Management Protocol, Version 3," RFC 3376 (Proposed Standard), Internet Engineering Task Force, Oct. 2002, updated by RFC 4604. [Online]. Available: http://www.ietf.org/rfc/rfc3376.txt

[4] H. Holbrook, B. Cain, and B. Haberman, "Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast," RFC 4604 (Proposed Standard), Internet Engineering Task Force, Aug. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4604.txt

[5] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, *Scalable application layer multicast*. ACM, 2002, vol. 32, no. 4.

[6] D. A. Tran, K. A. Hua, and T. T. Do, "A peer-to-peer architecture for media streaming," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 121–133, 2004.

[7] I. Wijnands, E. Rosen, A. Dolganow, T. Przygienda, and S. Aldrin, "Multicast using bit index explicit replication (BIER)," RFC 8279 (Experimental), Internet Engineering Task Force, Nov. 2017. [Online]. Available: http://www.ietf.org/rfc/rfc8279.txt

[8] I. Wijnands, E. Rosen, A. Dolganow, J. Tantsura, S. Aldrin, and I. Meilik, "Encapsulation for bit index explicit replication (BIER) in MPLS and non-MPLS networks," RFC 8296 (Experimental), Internet Engineering Task Force, Jan. 2018. [Online]. Available: http://www.ietf.org/rfc/rfc8296.txt

[9] L. Ginsberg, A. Przygienda, S. Aldrin, and J. Zhang, "Bit index explicit replication (BIER) support via IS-IS," RFC 8401, Internet Engineering Task Force, Jun. 2018. [Online]. Available: http://www.ietf.org/rfc/rfc8401.txt

[10] R. Boivie, N. Feldman, Y. Imai, W. Livens, and D. Ooms, "Explicit Multicast (Xcast) Concepts and Options," RFC 5058 (Experimental), Internet Engineering Task Force, Nov. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5058.txt

[11] M. Hansen, R. Smith, and H. Tschofenig, "Privacy Terminology," Internet Engineering Task Force, Tech. Rep. draft-hansen-privacy-terminology-03, Oct. 2011, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-hansen-privacy-terminology-03

[12] W. Fenner, "Internet Group Management Protocol, Version 2," RFC 2236 (Proposed Standard), Internet Engineering Task Force, Nov. 1997, updated by RFC 3376. [Online]. Available: http://www.ietf.org/rfc/rfc2236.txt

[13] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP multicast service and architecture," *IEEE Network*, vol. 14, no. 1, pp. 78–88, 2000.

[14] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, "A survey of application-layer multicast protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 58–74, 2007.

[15] K. T. Phan, N. Thoai, E. Muramoto, K. Ettikan, B. Lim, and P. Tan, "Treemap-the fast routing convergence method for application layer multicast," in *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*. IEEE, 2010, pp. 1–5.

[16] K. T. Phan, J. Moulierac, C. Tran, and N. Thoai, "Xcast6 treemap islands: revisiting multicast model," in *StudentWorkshop@CoNEXT*, 2012.

[17] R. Hinden, "Internet protocol, version 6 (IPv6) specification," RFC 8200 (Best Current Practice), Internet Engineering Task Force, Jul. 2017. [Online]. Available: http://www.ietf.org/rfc/rfc8200.txt

[18] M. Waehlisch, T. Schmidt, and S. Venaas, "A Common API for Transparent Hybrid Multicast," RFC 7046 (Experimental), Internet Engineering Task Force, Dec. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7046.txt

[19] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *INFOCOM'96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, vol. 2. IEEE, 1996, pp. 594–602.

[20] Y.-h. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE Journal on selected areas in communications*, vol. 20, no. 8, pp. 1456–1471, 2002.

[21] J. McCann, S. Deering, J. Mogul, and R. Hinden, "Path MTU discovery for IP version 6," RFC 8201 (Best Current Practice), Internet Engineering Task Force, Jul. 2017. [Online]. Available: http://www.ietf.org/rfc/rfc8201.txt

[22] J. Abley, P. Savola, and G. Neville-Neil, "Deprecation of Type 0 Routing Headers in IPv6," RFC 5095 (Proposed Standard), Internet Engineering Task Force, Dec. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5095.txt

[23] F. L. Faucheur, "IP Router Alert Considerations and Usage," RFC 6398 (Best Current Practice), Internet Engineering Task Force, Oct. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6398.txt

[24] C. Partridge and A. Jackson, "IPv6 Router Alert Option," RFC 2711 (Proposed Standard), Internet Engineering Task Force, Oct. 1999, updated by RFC 6398. [Online]. Available: http://www.ietf.org/rfc/rfc2711.txt