INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

Evaluation of medium-sized networks based on an original implementation of the MEADcast Router

Adrian Schmidt

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

Evaluation of medium-sized networks based on an original implementation of the MEADcast Router

Adrian Schmidt

Aufgabensteller:	PD Dr. rer. nat. Vitalian Danciu
Betreuer:	Daniel Diefenthaler Fabian Dreer Cuong Tran
Abgabetermin:	18. April 2024

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 18. April 2024

.....

(Unterschrift des Kandidaten)

Abstract

Despite ever-increasing bandwidth demands and the compatibility of numerous internet services with multicast communication, the deployment of IP Multicast has not met expectations. Consequently, this thesis evaluates a novel multicast protocol named MEADcast, focusing on its feasibility, performance, and potential application domains. The evaluation utilizes a Linux Kernel implementation of the router along with a standalone implementation of the sender. Four distinct use cases, each representing real-world scenarios potentially benefiting from multicast communication, are examined. The investigation demonstrates the feasibility of deploying MEADcast in medium-sized networks, offering promising initial insights into the real-world applicability of MEADcast. MEADcast has proven its adaptivity and suitability for dynamic network environments, showcasing resilience and recovery capabilities in response to network disruptions, particularly attributable to its well-functioning fallback mechanism. MEADcast deployment results in significant performance improvements, including a 56% reduction in network bandwidth utilization, an 81.23% decrease in sender upstream bandwidth consumption, and a 49.18% reduction in total transfer time. Overhead generated by the discovery phase primarily originates from the initial discovery, with subsequent recurring discovery phases incurring negligible overhead. MEADcast is well-suited for both applications characterized by communication patterns of steady flows (e.g. multimedia stream) and recurring bursts (e.g. file transfer). The protocol excels in scenarios characterized by resource constraints and limited network control. Deploying a single MEADcast router at a strategic location can significantly reduce bandwidth utilization between the sender and router. However, increasing the number of MEADcast routers may not necessarily enhance performance due to packet replication inefficiencies. Given its resilient fallback mechanism and support for partial deployment, MEADcast adoption is advocated whenever multicast communication is applicable but IP Multicast is unavailable. Nonetheless, several refinements are proposed for the upcoming MEADcast revision, aiming to address existing inefficiencies and mitigate malicious interference.

Contents

1. Introduction

1.1. Motivation

Over the last two decades, the number of people with access to the internet has continuously increased at a rate of 5% per year, resulting in a current total of over 5 billion individual internet users [ITU23]. Furthermore, the unrelenting demand for data has led to an average growth in total bandwidth usage of 30% over the past five years. This development was further accelerated by the COVID-19 pandemic, as the increased reliance on remote work, online education, and digital entertainment surged the bandwidth consumption [Car21]. These growth rates are expected to remain high in the future. Especially Multimedia content like Video Streams, Conferences, and Online Games depict a major portion of the global internet traffic. Video traffic is accountable for more than half of the bandwidth consumed in 2020 [Car21]. Network Operators and Service Providers need to comply with the continuously rising demand.

Already in the late 1980s, Deering and Cheriton proposed a multicast extension to the IP protocol, to facilitate efficient multipoint communication (*1:m, m:n*) [DC90; Dee89]. Multicast offers the advantage, to greatly reduce the occupied bandwidth by condensing identical traffic into a single stream, targeted towards multiple recipients [Cai+02]. Routers may replicate packets of that stream at points where the paths towards the receivers diverge. Many of today's internet services, particularly those with high bandwidth demands, may benefit significantly from multicast delivery [RES06; TD18]. Besides technical benefits, the adoption of multicast communication could reduce the emissions caused by Communication Technology (CT). Several studies assert that networks are accountable for a major portion of today's CT emissions [AE15]. Furthermore, they forecast strongly increasing energy consumption by networks until 2030. Multicast communication has the potential to lower energy consumption and therefore emissions by promoting more efficient use of network resources and potentially reducing the need for additional infrastructure.

1.2. Challenges of Multicast

The application of IP Multicast has yielded mixed results. Practically all of today's devices support IP Multicast [RES06]. Furthermore, it is utilized on a local scale, and various protocols like IPv6 Neighbor Discovery [Sim+07], RIPv2 [Mal98], OSPF [Moy98] and mDNS [CK13] rely on multicast.

However, more than thirty years since the initial proposal of IP Multicast, its global deployment and usage still lags far behind expectations [Dio+00; RES06]. Despite the potential advantages of Multicast, the majority of internet traffic continues to rely on point-to-point communication, known as *unicast* [Zha+06]. There are several reasons for the limited usage of IP Multicast.

Feasibility Besides the aforementioned advantages, IP Multicast entails various technical obstacles. Compared to unicast, it exhibits increased technical complexity, requiring the interaction of various protocols on multiple network layers [RES06; Dio+00]. Furthermore, multipoint communication in general interferes with the application of today's widespread security mechanisms like encryption [RH03]. Moreover, IP Multicast is based on a fixed address space [Dee89; DH06], which has an insufficient number of addresses. This limitation makes it infeasible to map highly dynamic sessions, like conferences, onto this space [DT19]. Additionally, the protocol involves a complex routing procedure, which requires all routers along the path to maintain a per-session state [Dio+00; RES06]. Consequently, global IP Multicast availability is constrained, as successful packet delivery necessitates all routers to support the protocol. However, this scenario is unlikely since many commercial routers come preconfigured with IP Multicast disabled [Aru19]. Moreover, the following paragraph describes, why Network Operators are probably not willing to enable it.

Desirability So far, Network Operators and Internet Service Providers (ISPs) have shown limited efforts to deploy Multicast within their Administrative Domains [Dio+00; RES06; ST02]. The increased technical complexity of Multicast requires more extensive management efforts. Additionally, due to the complex routing procedure, infrastructure upgrades may be necessary. Despite its potential for significant bandwidth savings, ISPs seem to assess the deployment of IP Multicast as an unsuitable investment [RES06]. Another reason is the more complex pricing model of multipoint communication. Charging for multicast services is non-trivial compared to existing unicast billing [RES06]. On top of that, IP Multicast hampers Network Operators' ability to anticipate network load. Forecasting the number of replicas generated from a multicast packet, entering the Network Operators Administrative Domain, is unfeasible [Dio+00]. Combined with the limited security mechanisms of IP Multicast, this represents a vulnerability, potentially exploited for amplification attacks. This fact makes intra-domain IP Multicast even more unlikely.

Unless ISPs face pressure to expand their service offering, increasing multicast deployment is doubtful. As multicast delivery has no direct impact on receivers, customers are not expected to exert the necessary pressure. Moreover, statistics illustrate that more than half of the internet traffic volume is HTTP-based [Clo23], which is not well suited for multipoint communication. Additionally the usage of Multicast is discouraged by web browsers, the most widely utilized HTTP clients, due to their technical limitation to TCP/HTTP[1].

1.3. Goal and Contribution

The current state of the internet put forth various unicast-based alternatives, aimed at addressing the absence of a globally usable multicast protocol [Zha+06]. One such alternative, known as Multicast to Explicit Agnostic Destinations (MEADcast) [TD18], offers the capability for 1:n sender-based IPv6 multipoint communication over the internet [DT19]. Key features of MEADcast include the preservation of receiver privacy, technology-agnostic destinations, and zero network support requirements.

[1] Despite the growing popularity and browser support for QUIC [IET21; JC20], also known as HTTP over UDP, it encounters similar challenges as TCP/HTTPS. This include issues such as packet acknowledgment and the client side generation of random numbers [IT21].

Building upon prior research conducted by Danciu and Tran [DT19], the primary goal of this thesis is to conduct a evaluation of MEADcast, utilizing a Linux Kernel implementation of the required router software. This step represents a logical progression from earlier investigations of MEADcast, which were based on network simulations [TD18] and Software-defined networking (SDN) [Ngu19]. The evaluation primarily focuses on the following aspects:

Feasibility Study The first part of the evaluation assesses the feasibility of deploying MEADcast in a medium-sized network. This aims to identify potential limitations and structural issues of the current protocol specification.

Performance Evaluation A comparative performance analysis is conducted to evaluate MEADcast in comparison to both IP Unicast and multicast. This assessment provides insights into the efficiency and effectiveness of MEADcast as a multipoint communication solution.

Scenario identification Building on the results of the previous evaluation steps, this phase involves identifying scenarios, application categories, and characteristics that justify the utilization of MEADcast. Thereby, it can be determined where MEADcast may offer advantages and excel in real-world applications.

Aligned with the established objectives, this thesis presents a series of contributions. Firstly, it introduces a Linux Kernel implementation of the MEADcast router, facilitating the deployment of MEADcast in a real network. Second, it provides a standalone sender implementation, enabeling arbitrary applications to utilize MEADcast delivery by directing their traffic into a TUN device. These contributions are further solidified through the deployment of MEADcast in a medium-sized network. Lastly, this research evaluates MEADcast's feasibility, performance, and potential application scenarios based on a series of conducted use cases. This evaluation provides valuable insights for its future implementation and development, ultimately enabling us to propose a revision of the protocol specification.

The findings indicate, the feasibility of deploying MEADcast within medium-sized networks. Furthermore, MEADcast demonstrates significant reductions in network bandwidth utilization, sender upstream bandwidth consumption, and total transfer time compared to IP Unicast. Nonetheless, the current specification exhibits from routing inefficiencies, header overhead, and security concerns. To address these issues, this thesis proposes several refinements. Our measurements suggest that MEADcast's performance falls between uni- and multicast, particularly showing promise in scenarios characterized by limited bandwidth and network control.

1.4. Method

To ensure the achievement of the previously defined goals, this thesis follows the procedure illustrated in Figure 1.1. First, we conduct a literature review of several multicast protocols and analyze the current challenges of multipoint communication. With a clear understanding of the relevant protocols and their limitations we define the objectives of this thesis. Subsequently, we explore application domains and characteristics potentially benefiting from multicast communication. Furthermore, we select adequate evaluation metrics and criteria

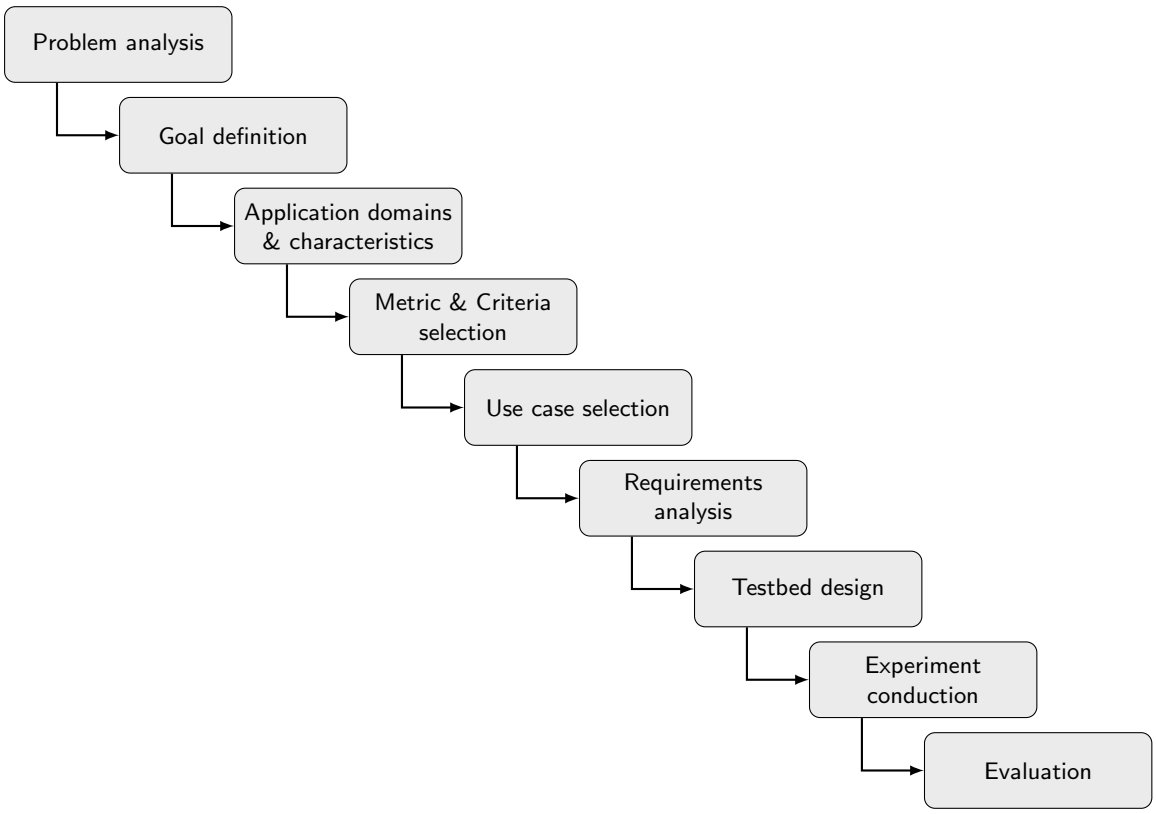

Figure 1.1.: Research method

to asses MEADcast's feasibility, performance, and potential application scenarios. Subsequently, we design a series of use cases aimed at capturing these metrics. Next, we outline the testbed requirements derived from these use cases and elaborate on a corresponding testbed design. MEADcast is then deployed in the testbed, and the experiments are conducted. Finally, we present the obtained results. These findings are critically evaluated with respect to the thesis' overarching goal, providing valuable insights into the feasibility and performance of MEADcast. This analysis allows us to identify scenarios and applications for which the protocol is well-suited.

1.5. Structure

The method employed in this thesis ensures a cohesive narrative, with each chapter building upon the knowledge acquired in the previous sections. Chapter 1 depicts the motivation for investigating multipoint communication, highlights the current challenges of IP Multicast, and articulates the goal of this thesis. Moving forward, Chapter 2 establishes the theoretical foundation for subsequent investigations, by examining several multicast protocols, including IP Multicast, Explicit Multicast (Xcast), Explicit Multicast Extension (Xcast+), and MEADcast. Given that, we have to design a representative medium-sized network topology, this chapter also provides a brief overview of a state-of-the-art model for medium to large-sized networks. Additionally, this chapter provides a brief overview of a state-of-the-art model for medium to large-sized network, laying the groundwork for the subsequent net-

work topology design. Chapter 3 presents a range of potential application domains, their characteristics, our selection of research questions and evaluation metrics, a series of use cases, testing parameters, and implementation requirements. Chapter 4 delves into technical details about the Kernel implementation of the router software, sender functionality, and provides detailed specifications of the topology and experiment conduction. This chapter serves as the foundation for the practical experiments. Moving forward to Chapter 5, we present the results from our experiments and evaluate them. Finally, Chapter 6 summarizes the findings and draws conclusions from this research. Additionally, it outlines potential avenues for future work and exploration in this field.

2. Background and Related Work

This chapter serves to establish the theoretical background for our study. Section 2.1 introduces various multicast protocols, including IP Multicast, Xcast, Xcast+, and MEADcast. Section 2.2 lays the groundwork for our topology design by presenting a state-of-the-art model tailored for designing medium to large-sized network topologies.

2.1. Multicast Protocols

Multicast communication refers to the simultaneous delivery of data towards an arbitrary number of destinations [KPP93; LN93]. Numerous network protocols have been developed to facilitate multicast communication, encompassing both Network Layer as well as Application Layer implementations [Zha+06; ST02]. This section first distinguishes Multicast from other communication schemes, followed by an introduction to various multicast protocols.

According to the OSI model [Zim80] Layer 3, known as the Network Layer, is responsible for end-to-end delivery of data between nodes. To establish a path from the sender to the destination(s), packets (Layer 3) may traverse multiple intermediate nodes [Pos81]. This process is called routing and can be classified into various schemes. For the purpose of this thesis, our primary focus lies on the schemes depicted in Figure 2.1. *Unicast* denotes a one-to-one association between a sender and a single destination. *Broadcast* disseminates packets to all nodes within the sender's broadcast domain (Layer 2) [Mog84]. Typically, IP Routers (Layer 3) serve as the boundary of a broadcast domain. *Multicast* transmits packets to a group of destinations, accommodating both one-to-many and many-to-many communication [Dee89]. In contrast to Broadcast, Multicast does not necessarily deliver packets to all available nodes. Furthermore, Multicast packets can be delivered beyond the sender's broadcast domain, implying subsequent replication of the packets.

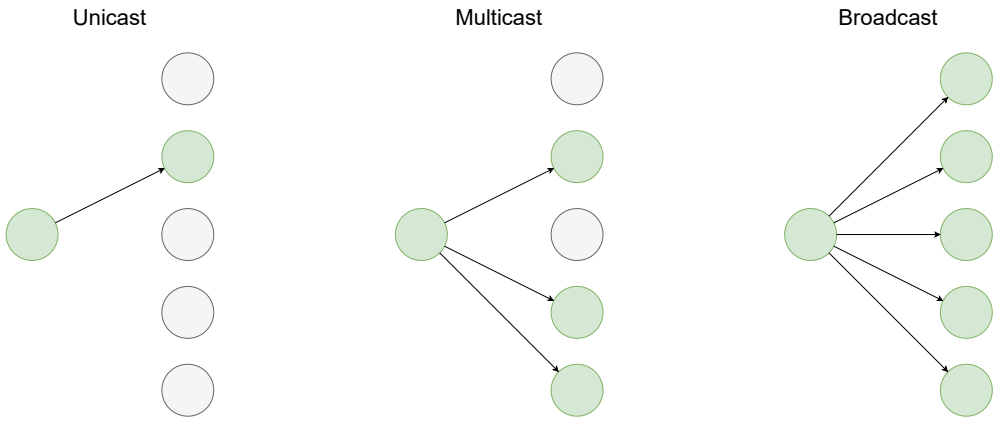

Figure 2.1.: Network Layer: Routing schemes

2.1.1. IP Multicast

Already in the late 1980s, Deering and Cheriton [DC90] proposed an extension to the IP protocol, to facilitate efficient multipoint communication (*1:n*, *m:n*). This extension is known as IP Multicast and was first standardized in RFC 988 (1986) [Dee86]. IP Multicast offers the advantage, to greatly reduce the bandwidth by condensing identical traffic into a single stream targeted to a so called *"Host Group"* [Dee89], which is identified by an IP multicast address. This characteristic facilitates high scalability, such that host groups may encompass thousands of receivers, since senders are not required of any knowledge about the number of receivers nor receivers identity or location [Cis01]. Intermediate nodes like IP routers replicate multicast packets addressed to an host group where the paths towards the receivers diverge.

Addressing IP Multicast operates on a separate address space and host groups are identified based on an IP address within the designated address range. IPv6 multicast addresses are defined within the range of `"fd00::/8"`, with the most significant octet being set to `"ff"`, indicating multicast addressing [DH06].

Group Management To join a host group, receivers express interest in receiving a particular data stream [Cis01]. In IPv4 networks Internet Group Management Protocol (IGMP) is used to establish host group membership [Cai+02], while group management in IPv6 networks is handled by Multicast Listener Discovery (MLD) [CV04], being embedded into Internet Control Message Protocol for IPv6 (ICMPv6) [GC06]. Hosts join IPv6 multicast groups either by sending an unsolicited MLD report or by responding to a general query from an IPv6 multicast router with an MLD report [Cis]. Similarly, when hosts wish to leave a multicast group, they have multiple options: they can disregard the periodic MLD general queries, a process referred to as a "silent leave", they can actively signal their departure by sending an MLD filter mode change record, or they can send so-called `"BLOCK_OLD_SOURCES"` messages with the IPv6 multicast address identifying the host group they want to leave. [Cis]. The latest standards IGMPv3 and MLDv2 enable receivers to filter traffic sources, a process referred to as Source-specific Multicast (SSM), by signaling their connected router from which unicast address it wants to receive traffic [Cis01], thereby hosts send a tuple of multicast and unicast address to the router.

Routing Multicast routers create *distribution trees* that determine the path multicast packets traverse through the network [Cis01]. The are two types of distribution trees: source and shared trees.

The most basic form of a multicast distribution tree is a source tree, where the root is located at the source and branches extend through the network to reach the receivers, forming a spanning tree [Cis01]. Given that this tree employs the shortest path through the network, is is often referred to as Shortest Path Tree (SPT).

In contrast, shared trees utilize a single common root positioned at a selected location within the network [Cis01]. This central shared root, is referred to as a Rendezvous Point (RP).

Forwarding In unicast routing, traffic follows a singular path from the source to the destination host within the network. A routers forwarding decision is solely based on the

destination address [Cis01]. Contradictory, in multicast forwarding, the source transmits data to a designated group of hosts identified by a multicast group address. At points of diverging path, routers replicate the packets and forward them accordingly. The process of directing multicast traffic away from the source, rather than toward the receiver, is referred to as Reverse Path Forwarding (RPF) [Cis01]. RPF stands as a pivotal concept in multicast forwarding, facilitating routers to accurately route multicast traffic along the distribution tree [Cis01].

Protocols There is a variety of multicast protocols, which can be classified into intra and inter-domain routing protocols. Common source tree based intra-domain routing protocols are DVMRP [88], MOSPF [Moy94], and PIM-DM [NAS05], while PIM-SM [Fen+16] is a common shared tree based protocol. Well known inter-domain routing protocols are MBGP [Cha+07] and MSDP [MF03].

2.1.2. Explicit Multicast

Traditional multicast protocols excel in scaling with large multicast groups but face challenges with a high number of distinct groups. Xcast, on the other hand, is a multicast protocol with complementary scaling properties compared to the traditional approach [Boi+07]. Xcast is designed to efficiently support a vast number of small multicast sessions by encoding receiver addresses explicitly in each packet instead of relying on multicast addresses [Boi+07]. Furthermore, Xcast eliminates the necessity for per-session signaling and per-session state information, which are characteristic of IP Multicast. The sender encodes the list of destinations in the Xcast header and transmits the packet to a router. Upon receiving the packet, each router along the path processes the header, partitions the destinations based on their next hop and forwards a packet with a corresponding Xcast header to each next hop [Boi+07]. When only one destination remains for a next hop, the Xcast packet can be transformed into an IP Unicast packet, a process referred to as Xcast to Unicast (X2U) [Boi+07].

In the example illustrated in Figure 2.2, the sender S transmits data to receivers E_1, E_2, and E_3. Consequently, S transmits a Xcast packet to router R_1: `{src=S, dst={E1, E2, E3}}`. Upon receiving an Xcast packet, a router process the header as follows: [Boi+07] (1) Perform a routing table lookup for each destination listed in the packet to determine their next hop. (2) Group the destinations based on their next hops. (3) Create a replica of the packet for each next hop. (4) Modify the destination list of each replica to inclide only the destinations that should be routed through that next hop. (5) Send each replica to its respective next hop. (6) If only one destination remains for a next hop, perform a X2U.

In the scenario demonstrated in Figure 2.2, router R_1 sends a single Xcast packet with a destination list of $\{E_1, E_2, E_3\}$ to R_2. Router R_2 sends a replica with a destination list of $\{E_2, E_3\}$ to R_4 and performs a X2U, transmitting an IP Unicast packet to R_3. R_3 forwards the IP Unicast packet as usual to receiver E_1. R_4 forwards the packet to router R_5 without creating any replicas. When the packet reaches R_5, X2U are performed for both E_2 and E_3, and the resulting packets are forwarded accordingly to R_6 and R_7. Finally, routers R_6 and R_7 forward default IP Unicast packets to E_2 and E_3, respectively.

For IPv6 Xcast is encoded in the routing extension header, optionally followed by a destination extension header specifying a port list, particularly in cases where receiver ports are diverging [Boi+07]. The IP destination field carries the *"All-Xcast-Routers"* multicast

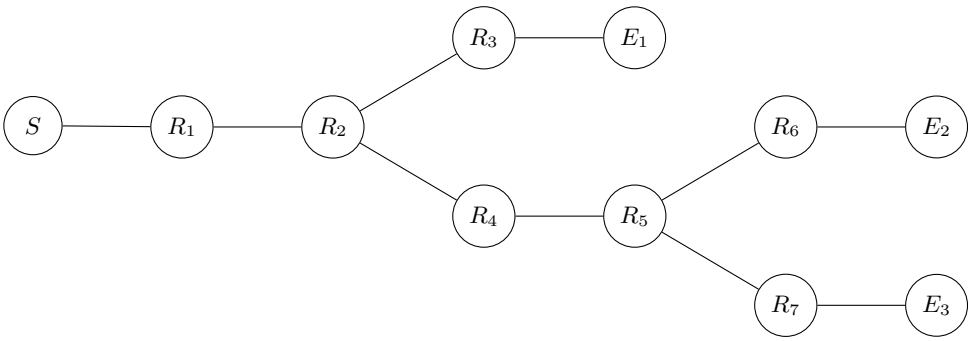

Figure 2.2.: Xcast processing (based on [Boi+07])

address, necessitating each Xcast router to join this multicast group. As a result, each router is required to support Xcast. For a gradual deployment, tunnel connections must be established between Xcast routers. Alternatively, a tunnel connection can assembled to one Xcast-aware receiver. This receiver has the capability to forward the packet either as an Xcast or IP Unicast packet to the remaining endpoints [Boi+07].

2.1.3. Explicit Multicast Extension

There are several extensions to Xcast, one of which is known as Xcast+ [Shi+01]. A key enhancement of Xcast+ is the reduction of the header overhead through the introduction of Designated Routers (DRs) [Shi+01]. The router nearest to a receiver determines itself as its DR. A client initiates the session by sending an IGMP (S, G) message (S: sender address, G: group address) to its DR. Upon receiving, the DR sends a so-called Xcast+ registration request message encompassing the sender's address (S), group address (G), and its own address towards the sender. When the DR at the sender's side receives it, it keeps track of all the DR addresses interested in the multicast session (S, G). When the sender transmits a multicast packet to its corresponding DR, it explicitly encodes the addresses of the DRs at the receivers' side in the Xcast header and transmits the Xcast+ packet, a process referred to as Multicast to Xcast (M2X). Consequently, DRs are not stateless anymore. Upon receiving Xcast packets, they perform a so-called Xcast to Multicast (X2M) transformation. As a result, the corresponding address list of Xcast+ might be significantly shorter compared to its Xcast alternative.

For instance in the scenario depicted in Figure 2.2, E_1, E_2, and E_3 send an IGMP join message (S, G) to their respective DRs R_3, R_6, and R_7. Upon receiving, the receivers' side DRs R_3, R_6, and R_7 send an Xcast+ registration request (S, G, R_i) towards sender S. When the DR on the sender side receives the Xcast+ registration requests, it adds a tuple for R_3, R_6, and R_7 encompassing (S, G, R_i) to its Xcast+ table. Upon receiving multicast packets from S, R_1 creates an Xcast packet with a destination list of $\{R_3, R_6, R_7\}$. The processing of the intermediate nodes is identical to Xcast, expect that no X2U transmission is taken place. When the client side DRs R_3, R_6, and R_7 receive the packet, they perform an X2M transformation, sending a multicast packet to E_1, E_2, and E_3 respectively.

2.1.4. MEADcast

MEADcast is a sender-centric multicast protocol designed to facilitate a seamless transition from extensive unicast to multicast transmission [DT19]. The sender is responsible for all group management tasks, while the receivers consistently receive IP Unicast traffic, remaining agnostic to the use of MEADcast [TD18]. MEADcast endures across varying levels of network support, allowing for a gradual deployment of MEADcast router. The protocol's design draws inspiration from Xcast [TD18].

MEADcast operates in two phases: discovery and data delivery. On the sender side, this entails functionalities such as transmitting unicast and MEADcast packets as well as discovering MEADcast routers along the paths to receivers [DT19]. MEADcast routers, on the other hand, are responsible for forwarding and transforming MEADcast packets, as well as responding to and forwarding discovery requests [DT19].

Figure 2.3 illustrates the interaction among the MEADcast sender, routers, and receivers: [DT19]

(1) A session is established between the sender and the receivers. MEADcast does not specify any mechanism for joining and leaving a session. Therefore, an out-of-band session establishment is required, which can be based on a predefined receiver list on the sender or initiated by a request from a client.

(2) The sender initiates data transmission. Due to the absence of a topology tree, data is transmitted via IP Unicast. MEADcast routers forward IP Unicast packets as usual.

(3) Concurrently with step (2), the sender starts the initial discovery phase by transmitting discovery requests to all receivers. Upon receiving discovery requests, a router sends a discovery response back to the sender and forwards the discovery request towards the designated receiver. The router reads the hop count from the discovery request and increments it for both for the forwarded request and the discovery response. If a client receives a discovery request, it ignores the packet.

(4) Upon receiving a discovery response, the sender inserts the router that sent the packet into its topology tree. The topology tree is a graph with the sender as the root and the receivers as leaves. MEADcast routers represent intermediate nodes, inserted at a position based on the hop count retrieved from the discovery response.

(5) After a predefined timeout is reached, the sender starts the data transmission phase by switching to MEADcast data delivery. Until this point in time, the IP Unicast transmission continued in parallel with the discovery phase. Receivers are grouped into MEADcast headers based on the current topology tree. Each MEADcast router along a packet's path processes it based on the information encoded in the MEADcast header. It might forward the packet, create replicas, and forward them to other MEADcast routers, or perform a MEADcast to Unicast (M2U) transformation, delivering the data to its destination.

Protocol Header MEADcast is a multicast protocol based on IPv6 Unicast traffic. In reference to Xcast, a list of receivers and their responsible MEADcast routers is encoded

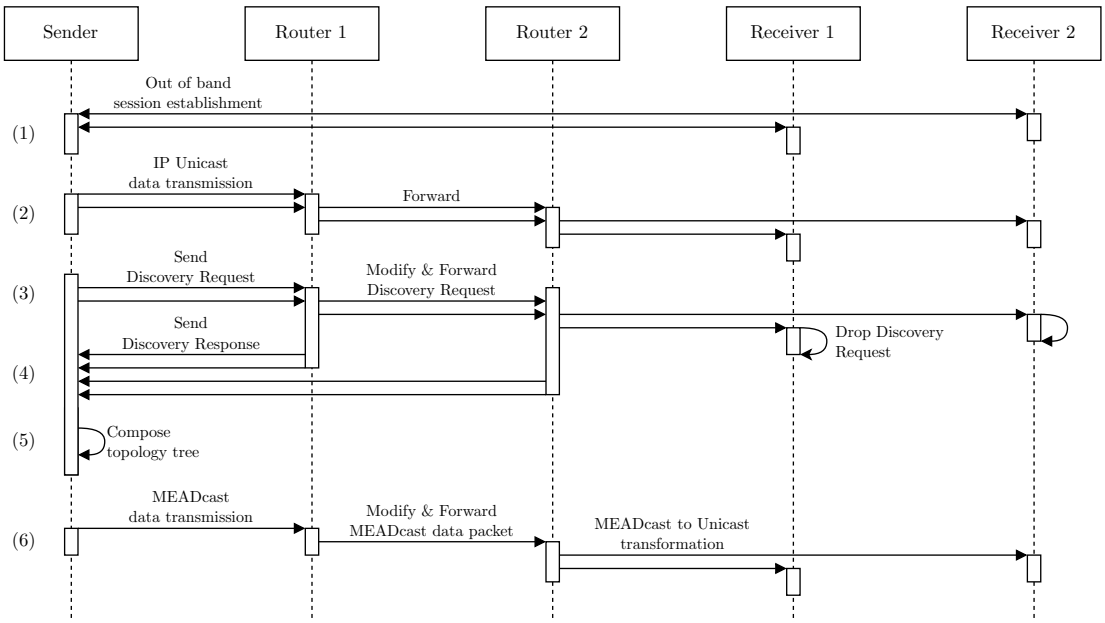

Figure 2.3.: Interaction among participants in a MEADcast session (based on [DT19])

in the IPv6 routing header extension [DT19]. The Routing Header follows an IPv6 Hop-by-Hop extension header containing the Router Alert option[DT19]. Optionally, a port list encoded in the IPv6 Destination Option header may follows the Routing Header. An optional port list encoded in the IPv6 destination option header may follows the Routing Header [DT19]. Figure 2.4 attempts to illustrate the MEADcast header. However, the papers about MEADcast [TD18; DT19] lack an explicit specification of the header fields. For instance, the content of the Router Alter Option Value, the Routing Type, and the Destination Option Header preamble are not specified. Additionally, the size of the discovery flags and the MEADcast hops fields are not defined. A brief description of each field is provided in Table 2.1.

Discovery Phase The sender sends discovery requests of the form $req(E_i, d)$ (E_i = receiver, d = distance) to all endpoints [TD18]. The routers respond to discovery requests with a disovery response $res(E_i, d, R_i)$ (R_i = router). Upon receiving a discovery response, the sender updates its topology tree in the following format: (R_j, d_j, E_1^j, E_2^j). For instance in the topology depicted in Figure 2.5 the discovery phase look as follows:

(1) Sender S start transmitting data to E_1, E_2, E_3, E_4, and E_5 via IP Unicast.

(2) S sends a discovery request to each receiver, initialized with a distance of zero: $req(E_i, 0), i \in 1, 2, 3, 4, 5$.

(3) Upon receiving the discovery requests, R_1 increments the hop counter and forwards the discovery requests: $req(E_i, 1), i \in 1, 2, 3, 4, 5$. Additionally, for each discovery request R_1 sends a discovery response back to S: $res(E_i, 1, R_1), i \in 1, 2, 3, 4, 5$.

(4) Step (3) is repeated on every router.

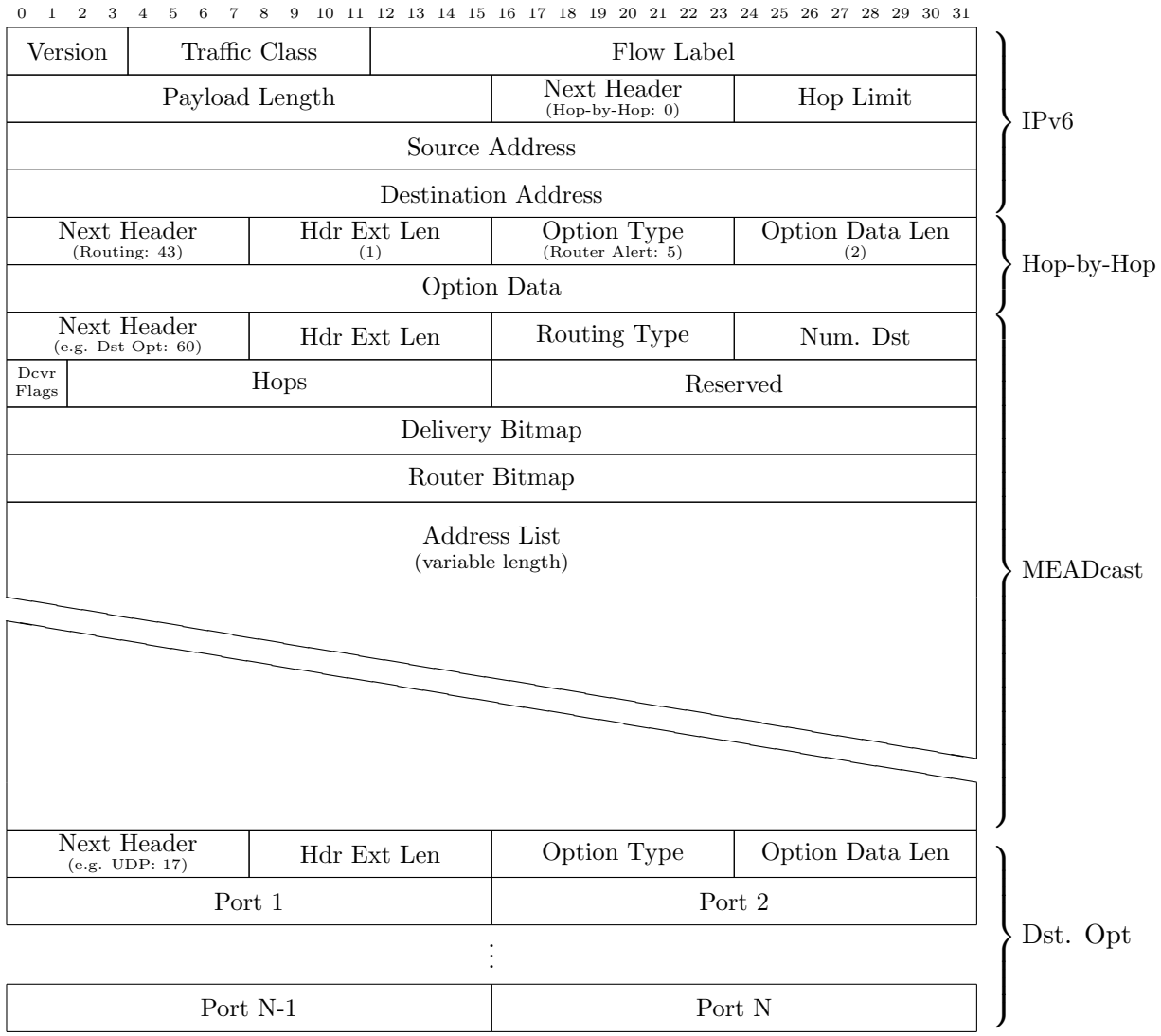

Figure 2.4.: MEADcast Header (based on [DT19; TD18])

Field	Desciption
IPv6 src addr	A 32-bit field, carrying the senders IPv6 address [DT19].
IPv6 dst addr	A 32-bit field, carrying the IPv6 address of one receiver [DT19].
Next Header	An 8-bit selector, identifying the type of the immediately following header [DH17]. It employs identical values as the IPv4 Protocol field (e.g. 17 = UDP, 59 = No Next Header for IPv6) [Aut24]. The Hop-by-Hop header field carries 43, indicating the following routing header. The Routing Header field carries either 60 in cases of the presence of the optional port list, and otherwise, it carries the number of the following Layer 3 protocol (e.g. 17 for UDP).
Hdr Ext Len	An 8-bit unsigned integer representing the length of the Routing header in 8-octet units, excluding the initial 8 octets [DH17].
Routing Type	An 8-bit identifier denoting a specific Routing header variant [DH17]. The MEADcast papers [TD18; DT19] lack a specification of this field.
Num. Dst.	Denotes the number of IPv6 addresses encoded in the address list. The size of this field lacks a specification.
Dcvr Flags	Flags to mark a MEADcast packet as discovery request, response or data message [DT19]. The size of the field lacks specification.
Hops	Specifies the distance between the sender and a router, denoted in the number of MEADcast hops. This field is utilized during the discovery phase. The sender initializes the field with zero, and it is incremented by each intermediate MEADcast router encountered. The field size lacks specification.
Reserved	A reserved field of unspecified size [DT19].
Delivery Map	A 32-bit Bitmap, where each bit at position i indicates whether a router at index i in the address list has already been delivered (0 = delivered, 1 = not yet delivered) [DT19].
Router Map	A 32-bit Bitmap, where each bit at position i indicates whether an address at index i in the address list is a router (0 = receiver, 1 = router) [DT19].
Address List	A variable-length list comprising the IPv6 addresses of receivers and MEADcast routers [DT19].
Dst Opt Preamble	The values for the Destination Option Preamble such as Option Type and Option Data Len lack specification.
Port List	A variable-length list encompassing the Layer 3 ports for each receiver in the address list [DT19].

Table 2.1.: MEADcast header field description

(5) Upon receiving discovery responses the clients ignore them.

(6) Upon receiving discovery responses S updates its topology tree finally resulting in the following topology: (R_1, 1, E_1, E_2, E_3, E_4, E_5), (R_2, 2, E_1, E_2, E_3, E_4, E_5), (R_3, 3, E_1), (R_4, 3, E_2, E_3, E_4, E_5), (R_5, 4, E_2, E_3), (R_6, 4, E_4, E_5).

(7) After a predefined timeout is reached, the sender stops transmitting data and switches to the data transmission phase.

Data Delivery During the data delivery phase, the receivers are grouped into MEADcast packets based on the topology tree constructed during the previous discovery phase [TD18]. The sender composes MEADcast data packet with an address list of the format $(R_j, E_1^j, E_2^j, \ldots, R_k, E_1^k, E_2^k, \ldots)$, with R_j being the closest MEADcast router to E_1^j and E_2^j and R_k being the closest router to E_1^k and E_2^j. The destination field of the IP header is set to the first receiver in the address list.

For the topology illustrated in Figure 2.5, sender S the data delivery proceeds as follows:

(1) Sender S transmits a MEADcast packet with an address list of $\{R_5, E_2, E_3, R_6, E_4, E_5\}$ and a destination IP address of E_2 to R_1. The router bitmap is initialized to `"100100"`, indicating that the addresses at index 0 and 3 are MEADcast routers and addresses at index 1, 2,4, and 5 are receivers. Moreover, the bitmap specifies, that R_5 is responsible for E_2 and E_3, and R_6 is responsible or E_4 and E_5. The delivery bitmap is initialized with the same value as the router bitmap, signaling no router has been delivered yet (1 = not delivered). During delivery, the router bitmap is never altered. Since R_3 is responsible only for one receiver, E_1 is served via IP Unicast, to save header space.

(2) Upon receiving the MEADcast data packet, R_1 parses the router and delivery bitmap. For each undelivered router, specified by a set bit in both bitmaps, the router checks whether it is one of its own IP addresses. If not, it creates a replica for each undelivered router. For each replica, the bits of all other routers are set to zero in the delivery bitmap, and the destination IP is set to the first address following the router address. In this scenario, R_1 parses the header and identifies an undelivered router address, which is R_5. Since it is not its own address, R_1 creates a replica for R_5. In the replica, the delivery bitmap is set to `"100000"`, and the destination IP is set to E_2. Subsequently, R_1 identifies an undelivered router at index 3. As it is the last router in the address list, no replica generation is necessary. Instead the original packet can be modified, and the delivery bitmap is set to `"000100"`. Finally, R_1 forwards the two MEADcast packets and the IP Unicast packet to R_2. Since MEADcast endures partial network support and R_1 lacks knowledge of whether there is another MEADcast router along the path, which could instead perform the packet replication for R_5 and R_6. For instance, R_2, R_3, and R_4 could be non-MEADcast routers. Consequently, R_1 sends separate replicas for R_5 and R_6, despite their shared path of MEADcast routers until R_4.

(3) Upon receiving the MEADcast packets, R_2 parses the header of both packets. Since the only router address in each packet is not owned by R_2, it just forwards both packets. The IP Unicast packet to E_1 is forwarded as usual. When R_3 receives the unicast packet, it delivers it to E_1.

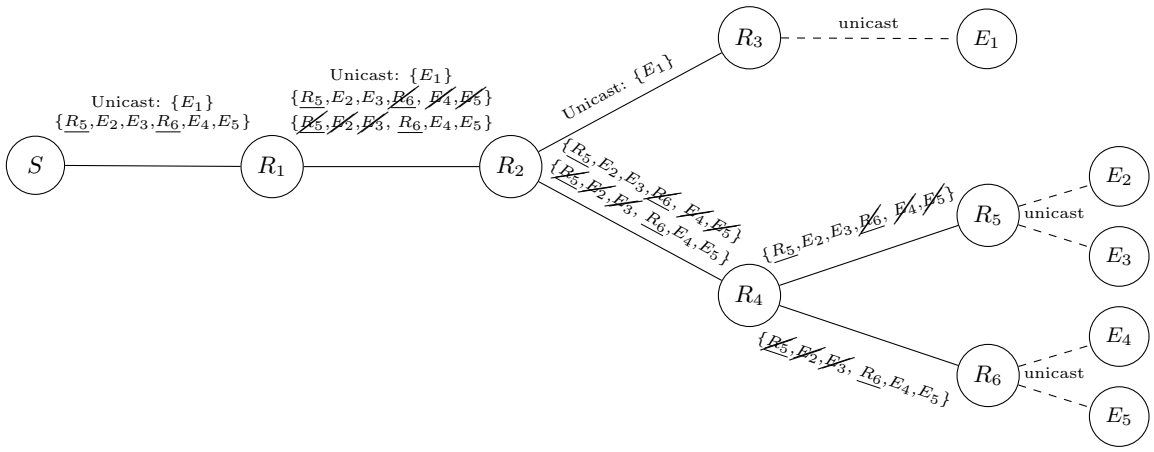

Figure 2.5.: MEADcast Data Delivery: An underscored address signifies a set bit in the router bitmap, while a stroked-through address indicates that the corresponding router bit is set to "delivered" (0) in the delivery bitmap.

(4) Upon receiving the MEADcast packets, R_4 parses the header of both packets. Since the only router address in each packet is not owned by R_4, it just forwards both packets to R_5 and R_6 respectively.

(5) Upon receiving the MEADcast packet, R_5 and R_6 parse the header, identifying their own address as the only undelivered router in the address list. Consequently, both routers perform an M2U transformation for each address following their own address. This involves creating an IP packet for each receiver, with the destination address set to the receiver's address. If the MEADcast packet includes the optional port list, the port in the Layer 3 header is set to the respective port from the list. If the Layer 3 protocol encompasses a checksum, it must be corrected, due to the alteration of the IP pseudo-header. For R_5, this process continues until another router address is encountered at index 3, while for R_6, it continues until the last index of the list. Subsequently, they each send IP Unicast packets to their respective destinations: E_2 and E_3 for R_5, and E_4 and E_5 for R_6.

2.2. Hierachical Three-Layer Internetworking Model

Since, we this thesis we design a medium-sized network topology, this section introduces state-of-the-art characteristics of medium to large-sized network architectures.

Several fundamental design principles must be met by any network, irrespective of its size [Aca14]. These principles are mutually interdependent and also tightly coupled to the overall design [Cis08].

Hierarchy A hierarchical network model serves as a crucial high-level tool for crafting a reliable network topology [Aca14]. This abstraction dissects the complex challenge of network design into smaller and more manageable areas. A significant advantage of a hierarchical topology is that local traffic is confined to the local network [Cis08]. Only traffic destined for other networks is transferred to a higher layer.

Modularity Dividing the network into components provides numerous benefits. First, each component or module can be designed with a level of independence from the overall structure [Aca14]. All modules can operate as semi-independent elements, contributing to higher overall system availability and facilitating simpler management and operations [Cis08]. Furthermore, network changes and upgrades can be applied in a controlled and staged manner, enhancing flexibility in maintaining and operating the network [Aca14]. This principle has also been leveraged in software architecture for many years, by separating the application into presentation, logic, and data layers [Ric15]. Another benefit is a higher degree of isolation, confining problems and failures to their component, leaving the overall network unaffected [Cis14]. In summary, dividing a network into a set of assembled building blocks facilitates increased stability, flexibility, and manageability for both the individual components and the network as a whole. [Cis08]

Resiliency The network is expected to sustain availability for both regular and irregular scenarios. Typical scenarios encompass anticipated traffic flows, expected traffic patterns, and scheduled events like maintenance windows [Aca14]. Abnormal scenarios involve hardware or software failures, high traffic loads, anomalous traffic patterns, denial-of-service (DoS) events, and other unforeseen circumstances [Cis08].

Flexibility As the operational lifespan of networks extends, it becomes imperative for their design to facilitate enhanced adaptability and flexibility [Cis08]. Adapting to evolving business environments and their associated communication requirements is a practical business and operational necessity. Flexibility is the capacity to modify specific sections of the network, introduce new services, or enhance capacity without necessitating a substantial overhaul or replacement of major hardware devices [Aca14].

To meet these ubiquitous network requirements, Haviland [Hav98] first proposed the *hierarchical three-layer internetworking model* in 1998. This model has evolved into an industry-wide standard for designing reliable, scalable, and cost-efficient networks [Aca14]. Leading networking and telecommunication providers like Cisco and Huawei recommend the implementation of the hierarchical three-layer internetworking model. [Cis08; Hua23]. The model divides networks into three layers: core, distribution, and access as illustrated in Figure 2.6 [Hav98].

Access Layer The access layer facilitates network access for end devices, encompassing PCs, smartphones, printers, and IP cameras [Aca14]. Operating commonly at layer 2, it establishes connectivity between end devices. Given the diverse range of end devices, this layer incorporates an array of network devices such as switches, access points, and routers [Cis08]. Serving as the demarcation point between the network infrastructure and computing devices [Cis08], the access layer acts as the initial security boundary, safeguarding other users, application resources, and the network itself [Cis14].

Distribution Layer The distribution layer aggregates traffic from multiple access layer devices and provides connectivity to the rest of the network by transmitting packets to other access layer devices or the core layer [Cis08]. In a typical scenario, connected routers at the distribution layer aggregate data originating from a building or a floor and establish connections to the core layer. For instance, in Figure 2.6, the two interconnected routers on

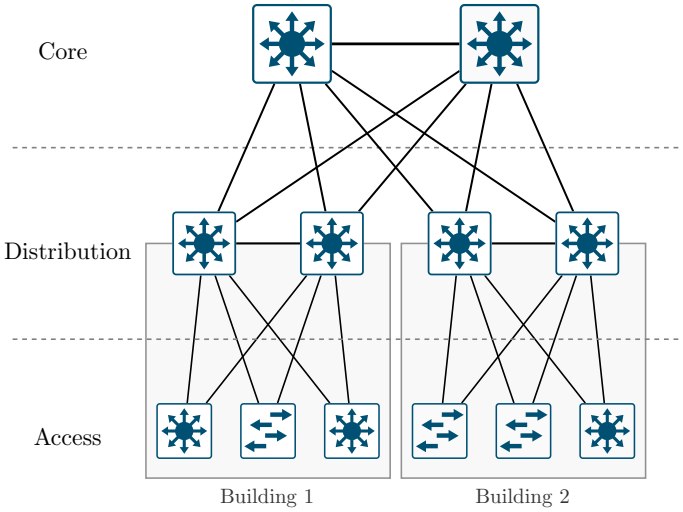

Figure 2.6.: Hierarchical three-layer internetworking model (based on [Hav98])

the left and the two on the right could each symbolize a distinct building. This layer often serves as the demarcation between layer 2 domains and the layer 3 routed network [Aca14], creating fault domains containing failures and network changes to the directly affected areas [Cis14]. The distribution layer performs essential network functions such as routing, traffic filtering, and policy enforcement [Cis08].

Core Layer Core LayerThe core layer functions as the backbone interconnecting the numerous modules of the network [Cis08]. It facilitates connectivity among end devices, servers, data storage, different locations, and the internet. This layer represents the most critical component of the network, characterized by a relatively simple design [Cis14]. Given that a significant portion of network traffic passes through the backbone, it is responsible for high-speed and high-volume data transmission [Cis08]. Additionally, this layer should offer high availability and redundancy. CPU-intensive packet manipulations resulting from security measures, inspections, and Quality of Service (QoS) should be avoided [Cis08].

3. Experiment Design

This chapter delves into the design of a series of use cases to evaluate MEADcast. To ensure, an comprehensive examination of MEADcast across a diverse range of use cases Section 3.1 initially elaborates on application domains that may benefit from multicast communication. Subsequently, Section 3.2 discusses the specific composition of characteristics that distinguish these domains. To maintain the quality of our study, Section 3.3 discusses research questions and measurements derived from the previously elaborated application domains and characteristics, guiding our research process. Following this, a selection of specific use cases is presented in Section 3.4, aimed at addressing our research questions. Next Section 3.5 introduces several testing parameters shared across all use cases. Finally, Section 3.6 elaborates on the requirements for our implementation, specifically regarding the topology, sender, and router software.

3.1. Application Domains

As highlighted in the Section 1.1, numerous motivations support the adoption of multicast communication. Additionally, various types of applications have the potential to benefit from multicast communication. However, it is essential to acknowledge that multicast is tailored for a specific form of communication – *simultaneous* transmission of *identical* data to multiple recipients, as delineated in Table 3.1. This inherent characteristic finds an ideal match in TV broadcasts, establishing them as a prime candidate for multicast communication. Conversely, Video on Demand services such as YouTube present a less favorable scenario for multicast adoption, as users request videos at different points in time. The advantages of multicast become particularly evident in bandwidth-intense applications, where its potential to significantly reduce the total communication volume is most pronounced. Real-time multimedia services therefore emerge as a particularly well-suited domain for the application of multicast. In the following, we introduce the formulated application domains, illustrating them with specific examples as detailed in Table 3.2. The characteristics distinguishing these domains are discussed in the following section.

	Temporal	
Content	Synchronous	Asynchronous
Identical	TV broadcast (yes)	Video on demand (no)
Different	- (no)	Web browsing (no)

Table 3.1.: Suitability of communication patterns for Multicast

The first application domain is *Multimedia Streaming*, encompassing various applications transmitting multimedia content to an arbitrary number of destinations, such as IPTV [DT19; RES06], Internet Radio [TD18], podcasts, and live streaming (e.g. Twitch[1]). *Conferencing and Collaboration* is the second domain, comprising Audio and Video Conferences [ST02; DT19; KPP93], Voice over IP (VoIP) [BGC04; Boi+07], as well as real-time collaboration applications [Dio+00; Boi+07] like online Mind Maps and Whiteboards. The next domain *File Transfer*, covers numerous applications distributing files to multiple recipients, including Software Distribution and Updates, Patch Management [TD18; RES06], Logging [Dio+00] as well as file sharing and synchronization [ST02]. *Information delivery* comprises applications pushing information to multiple destinations. One example is a news application, which notifies its users about new information of topics or channels they have subscribed [Dio+00]. Other applications include widely utilized smartphone notifications, RSS feeds [RES06], Logging (e.g. SNMP), and Stock quotes [Cis01]. The last domain is *Distributed Simulation*, with exemplary applications such physics simulations [Dio+00], virtual reality, and online gaming [RES06].

Domain	**Applications**
Multimedia streaming	IPTV, Internet Radio, podcasts, streaming platforms
Conferencing & Collaboration	Audio- & Video-Conferences, VoIP, Mindmaps, Whiteboards, ...
File Transfer	Software Distribution, Updates, Patch Management, Filesharing and -synchronization, Logging
Push notifications	RSS-feed, Logging, Stock quotes
Distributed simulation	Online Gaming, Virtual World, Simulation

Table 3.2.: Multicast application domains

3.2. Application Characteristics

This section delineates various characteristics derived from the application domains, categorized into three groups, as detailed in Table 3.3.

Group Group communication occurs in diverse forms. The number of participants in a multicast group displays substantial variation, ranging from small (< 10), to medium-sized and up to large-sized (> 100) groups. Additionally, the session duration exhibits significant diversity, spanning from a few minutes up to several days. The group *membership* can either be static or dynamic. For instance, small to medium-sized conferences might last several minutes to a few hours with mostly static group membership. In contrast, broadcasts of music or sports events endure for several days, attracting millions of viewers who may join or leave at any time.

[1] `https://www.twitch.tv/`

Domain	Group/Session			Communication		Network		
	Duration[1]	Mem. ship	Size[2]	Pattern	Interval	Pkt. size	Through. (tx/rx)	Latency /Jitter
Multimedia	h-d	dyn.	l	1:n	steady	med.	high/med.	med.
Conference	m-h	stat.	s-m	m:n	steady	med.	high/high	low
File transfer	s-h	stat.	s-l	1:n	burst	large	high/high	high
Push info.	s	stat.	m-l	1:n	burst	small	med./low	med.
Dist. sim.	m-h	stat.	s-m	m:n	steady	small	low/med.	low

[1] *(s)econds, (h)ours, (d)ays*
[2] *(s)mall, (m)edium, (l)arge*

Table 3.3.: Characteristics of the application domains (based on [Car21; Dio+00])

Communication The communication *pattern* within a group can either be symmetric (m:n) or asymmetric (1:n). Furthermore, the *interval* during which participants exchange messages may constitute a steady flow (e.g. video stream) or a recurring burst (e.g. file transfer). In a Peer-to-Peer (P2P) video conference or online game, all group members continuously transmit data to each other. Conversely, in an RSS feed, a single server periodically pushes information to multiple subscribers.

Network The network requirements of different applications exhibit significant variation. Specific applications exhibit sensitivity to distinct factors, with some prioritizing low *latency* and *jitter*, others demanding high *throughput*, and yet others being sensitive to the packet *drop rate*. For instance, many online games rely on low latency while maintaining frugal bandwidth requirements [Har+07]. In contrast, ensuring a high-quality video stream necessitates elevated throughput, even though increased latency is acceptable due to prevalent client-side buffering. Furthermore, an RSS feed has low throughput and moderate latency requirements. Nevertheless, it is sensitive to the packet drop rate, since it has to ensure successful data delivery.

3.3. Measurements

Aligned with the goal of this thesis, this section formulates overarching research questions derived from the previously introduced potential application domains, guiding the design of the use cases. Additionally, this section outlines our approach to answering these questions.

RQ1 How robust is the current MEADcast specification?

While prior research on MEADcast has primarily taken place in simulated environments [TD18; DT19] and small stub networks [Ngu19], this thesis endeavors to assess MEADcast in a more realistic setting. To achieve this, we perform a stress test to evaluate the protocol's behavior in a less "clinical" environment. During this test we observe MEADcast's reaction to deliberate routing changes, to evaluate its adaptivity and suitability for dynamic network environments. Further, we simulate network disruptions by inducing router outages, to assess MEADcast's resilience and recovery

capabilities. The stress test also examines MEADcast anomaly handling by injecting modified discovery responses. Additionally, a firewall is employed to intentionally drop MEADcast packets during the data delivery phase, enabling us to evaluate the efficacy of the fallback mechanism. The results of the stress test shed light on the robustness of the current MEADcast specification, especially in diverse network conditions. This investigation aims to provide valuable insights into the real-world applicability and resilience of MEADcast.

RQ2 How does MEADcast perform compared to IP Unicast and IP Multicast?

In addition to assessing the robustness of a protocol, its performance, especially in comparison to existing alternatives, is a pivotal factor influencing its adoption. To address this research question, comparative performance measurements for MEADcast, IP Unicast, and IP Multicast are conducted across a series of use cases elaborated in Section 3.4. These measurements encompass metrics such as throughput, latency, jitter, and resource utilization. Special attention is given to the impact of MEADcast's discovery phase on its performance. This encompasses assessing the overhead produced by the discovery mechanism. Furthermore, we evaluate how the protocol's shift from extensive unicast to MEADcast delivery affects the performance metrics. The outcomes contribute not only to evaluating MEADcast performance but also to drawing conclusions for subsequent research questions.

RQ3 Which applications and characteristics are well served by MEADcast?

Addressing this research question involves designing a series of use cases depicting a diverse range of applications with distinct characteristics (see Section 3.4). The selection of scenarios is guided by an initial exploration of motivations for employing multicast communication, and more specifically, MEADcast. These motivations lay the groundwork for identifying a range of potential application domains where MEADcast deployment could prove beneficial. Subsequently, we formulate application characteristics distinguishing these domains. Based on the gathered insights, various scenarios are chosen to represent different application domains and their unique set of characteristics. This comprehensive selection ensures a throughout examination of MEADcast's suitability across a diverse spectrum of use cases. The results aim to delineate MEADcast's application space by identifying its strengths and limitations.

RQ4 In which conditions is the usage of MEADcast sensible?

The viability of employing MEADcast is influenced not only by application domains and their characteristics but also by prevailing circumstances. To investigate this aspect, the experiments are executed with various parameter configurations. For instance, the level of network control and the number of available MEADcast routers affect the performance and thus the viability of MEADcast. This applies equally to testing parameters like available bandwidth, number of endpoints, and their distribution. The goal is to provide insights into the conditions under which deploying MEADcast is advantageous compared to existing alternatives. The results aim to offer guidance for making informed decisions regarding the adoption of MEADcast in specific circumstances.

3.4. Use cases

Based on the previously exhibited application domains and characteristics this chapter formulates several use cases.

The primary goal is, to develop a series of use cases encompassing various domains and characteristics, aimed at answering our research questions. This comprehensive selection ensures a throughout examination of MEADcast's suitability across a diverse spectrum of use cases. Therefore, the results from the use cases are pivotal to answer *RQ3*. The specific characteristics of each corresponding application domain are illustrated in Table 3.3. The chosen implementation for each use case is provided in Subsection 4.4.1.

UC1: Live Stream The first use case involves a video live stream, representing the multimedia stream domain. As reported by Cartesian [Car21], multimedia traffic is accountable for more than half of the bandwidth consumed in 2020. Therefore, this experiment is indicative of a major application space, with the potential for significant bandwidth savings. A conceivable scenario involves organizations such as universities or companies streaming events across multiple buildings on a campus or even across various locations. Typical campus layouts encompass both *local clustering* at certain offices or areas like the library, as well as a *low receiver density* for remote offices, resulting in *varying distances* between sender and receivers. Live streams usually endure *multiple hours*, attracting a *large number* of viewers who may *join or leave at any time*. Additionally, a common requirement for HD video streams is a *downstream bandwidth of 5 Mbit/s* [Car21]. This use case is well-suited to showcase MEADcast's suitability for long-running constant data streams, highlighting the effects of different receiver distributions. Given that multimedia streams generate both high traffic volume both at the sender and the network, it is of particular interest to explore the extend to which MEADcast can reduce the occupied sender upstream and network bandwidth. Moreover, we examine whether this reduction is accompanied by trade-offs such as increased latency/jitter or higher CPU utilization at the sender.

UC2: File Transfer The second use case emulates a file distribution, thus representing the file transfer domain. Especially scenarios such as software updates and file backups share major characteristics with this use case. Our use case simulates the distribution of a software update for a server Operating System (OS). Servers are usually located within a single network domain, leading to high receiver clustering. Additionally, updates are commonly obtained from an external source, often residing in another network domain. A software update is characterized by the utilization of all available downstream bandwidth on the receivers for a limited duration. Although mechanisms exist to limit bandwidth consumption, we only consider scenarios of full downstream bandwidth utilization. The number of receivers varies significantly, ranging from a few servers to entire data centers. The primary objective of this experiment is to investigate how well MEADcast handles recurring bursts of high traffic volume. Additionally, we aim to asses the reduction in total network bandwidth consumption, sender upstream bandwidth consumption, and total transfer time compared to IP Unicast. The characteristics of this use case are predestined to highlight MEADcast's performance under ideal conditions.

UC3: P2P Video Conference The third use case simulates a P2P video conference. Video Conferences commonly involve small to medium-sized groups and require a steady commu-

nication of 1-3 Mbit/s downstream and 0.5-1 Mbit/s upstream bandwidth [Car21], lasting from several minutes to a few hours. Our use case represents a video conference commonly seen in modern distributed work culture, involving locally clustered sub-teams (e.g. office) and some isolated participants (e.g. customer or service provider), with varying distances between sender and receivers. Furthermore, the inherent characteristic of P2P communication facilitates an environment in which the effects of high MEADcast traffic volume on MEADcast routers can be examined. This experiment is designed to investigate, whether MEADcast is capable of bridging asymmetric access link bandwidth [Boi+07; Car21]. Additionally, we examine the suitability of MEADcast for P2P communication. Lastly, we evaluate MEADcast's network bandwidth and sender upstream bandwidth utilization for small to medium-sized groups in comparison to IP Unicast and IP Multicast.

UC4: Online Gaming The last use case is an online multiplayer game, representing the application domain distributed simulation. Online games commonly encompass small to medium-sized groups, and require a steady downstream bandwidth of 0.5-1 Mbit/s [Car21], enduring for several minutes to a few hours. Most online games are highly sensitive to Latency and Jitter. To asses MEADcast's implications on these metrics, multiple MEADcast hops are essential. Consequently, this experiment has two primary objectives. First, to measure MEADcast's effect on latency and jitter. Secondly, to evaulate MEADcast's performance in suboptimal settings, particularly with a high receiver distribution.

3.5. Testing parameters

The series of use cases introduced earlier shares several testing parameters. Consequently, these scenarios incorporate diverse parameter configurations such as the number of MEADcast router, intended to assess their influence on the results from our experiments, specifically addressing *RQ2* and *RQ4*.

The first parameter is the *number and distribution of receivers*. Generally, with a larger number of receivers, a multicast protocol tends to achieve proportionally higher bandwidth savings compared to unicast. However, we anticipate that the distribution of receivers significantly impacts MEADcast's performance. Therefore, the use cases encompass various distributions, ranging from highly clustered receivers with minimal internal distances to uniformly spread distributions. This parameter aims to illuminate how spatial receiver arrangement influences the protocol's efficiency.

The second testing parameter is the *degree of MEADcast support*. Analogous to the first parameter, we hypothesize that the number and placement of MEADcast-capable routers have a significant impact on the protocol's performance. To investigate this, the experiments are conducted ranging from a minimal degree of MEADcast routers to complete protocol support within the testbed. By varying this parameter, we aim to assess MEADcast's efficiency under conditions of limited network control, offering valuable insights directly addressing *RQ4*.

The third, parameter resolves around *grouping receivers* into MEADcast packets. Evaluating the balance between the number of recipients per packet and the available Service Data Unit (SDU) size is crucial. A larger number of recipients per packet potentially reduces traffic volume by consolidating identical packet streams. However, this also enlarges the MEADcast header, consequently diminishing the available SDU size. This trade-off might

inadvertently increase traffic volume since more packets are required to transmit the total payload. Furthermore, the grouping algorithm itself is considered. For instance, recipients sharing the same parent router could be distributed across different packets to achieve an optimal balance between the MEADcast header and payload size. Moreover, receivers with distinct parents might be merged under a common ancestor to accommodate them within a single packet.

Lastly, the effect of different values for the *discovery interval* is also examined. Shorter intervals facilitate quicker adaption to topology changes, potentially leading to enhanced receiver grouping and faster error recovery. However, shorter intervals also result in higher traffic volume, which may impede data delivery.

3.6. Requirement analysis

Expanding upon on the formulated measurements and use cases, this section provides detailed insights into the requirements for our implementation. Subsection 3.6.1 discusses the requirements for the network topology necessary for conducting our experiment, while Subsection 3.6.2 examines requirements for the router and sender implementation.

3.6.1. Topology

Initially, the topology must transcend basic structures like plain bus or simple ring setups, ensuring a *non-trivial connected graph*. A graph is termed connected when there exists a path between every pair of vertices, representing clients and routers in this context. In order to thoroughly evaluate a multicast protocol, the topology needs to encompass a substantial number of nodes. Therefore, a *medium-sized topology* comprising around 200 nodes [Aca14], featuring a proportional mix of clients and routers, is pivotal.

Realistic topology Aligned with *RQ1*, the graph should reflect a realistic medium-sized network topology. As discussed in Section 2.2, a realistic topology comprises three network layers: Core, Distribution, and Access Layer. The Core Layer interconnects different network domains, facilitating high-speed and high-volume data transmission. Additionally, this layer is characterized by a relatively simple design and should offer high availability and redundancy. The Distribution Layer is required to aggregate traffic from multiple Access Layers and to provide connectivity to the rest of the network by transmitting packets to the core layer. The Access Layer layer is required to facilitate network access for end devices, encompassing PCs and smartphones. This layer must establish connectivity between end devices and serves as the demarcation point between the network infrastructure and computing devices.

Endpoint distribution The use cases require various numbers of endpoints and their distribution. The topology must accommodate group sizes ranging from small to medium-sized (UC3, UC4) up to medium to large-sized groups (UC1, UC2). Moreover, UC1, UC2, and UC3 require areas of *high receiver clustering*. Additionally, UC1, UC3, and UC4 mandate areas characterized by *low receiver density*. Since UC1 and UC3 require both local clustering and low receiver density, the topology must facilitate *varying distances* between sender and receivers. All use cases require the integration of *multiple network domains*. UC1,

UC3, and UC4 employ isolated remote hosts, necessitating the incorporation of paths of *sufficient length*. Additionally, the comprehensive evaluation of MEADcast's performance across diverse levels of network support and various grouping strategies, also requires paths of sufficient length, allowing packets to traverse multiple MEADcast routers for a thorough assessment, particularly of latency and jitter.

Flexibility Another crucial requirement is the flexibility of the topology. Diverse use cases and network conditions demand a modular and adaptable topology. Various placements of senders, receivers, and routers are necessary. Moreover, the use cases encompass receiver distributions ranging from highly clustered to uniformly spread. Additionally, the ability to adjust the quantity of MEADcast-capable routers is crucial for evaluating various levels of MEADcast support.

Connectivity Finally, several prerequisites pertain to the connectivity of the topology. Given the evaluation of a Layer 3 IPv6-based protocol, the presence of multiple subnets and IP routing is mandatory. Beyond that, IP Multicast routing is indispensable for a comparison of MEADcast and IP Multicast. Furthermore, our use cases mandate various up and downstream capacities. UC3 and UC4 require moderate downstream requirements of 1-3 Mbit/s and 0.5-1 Mbit/s, respectively. UC1 and UC2 demand a downstream bandwidth of at least 5 Mbit/s. Moreover, UC1, UC2, and UC3 also mandate an upstream bandwidth of multiple Mbit/s on the sender side. Additionally, assessing the resilience of MEADcast in dynamic network environments necessitates the inclusion of *alternative paths* to simulate routing alterations and network disruptions. Furthermore, to pursue *RQ1* the incorporation of an intermediate node, capable dropping packets is required.

3.6.2. Software

This section discusses requirements for the router and sender software implementation.

Sender To ensure a comprehensive assessment of MEADcast, the sender should include the following features:

First, the sender must periodically dispatch discovery requests to the receivers. Additionally, the sender needs to receive discovery responses and process them. After, the discovery timeout is exceeded the sender should start grouping the endpoints. Therefore, the sender is required to efficiently organize and group receivers into MEADcast packets, facilitating streamlined data transmission. Whenever possible, the software should support MEADcast data transmission. However, unicast is used as a fallback mechanism, if MEADcast is not available on certain paths. The discovery interval, discovery timeout, and grouping functionalities should be configurable, allowing flexibility for experimentation and optimization based on various network conditions and scenarios. Considering our evaluation of MEADcast is based on metrics such as latency and jitter, the sender should possess the capability to either measure these metrics directly or act as a proxy for traffic received from network measurement tools.

Router To enable the usage of MEADcast the router should encompass the following features:

The router should receive discovery requests, reply to them with a discovery response, and forward the request according to the destination address. Additionally, the router needs to receive, process, modify, replicate, and forward MEADcast data packets accordingly. For each receiver listed in the MEADcast header assigned to the router itself, the data should be transmitted the data to the respective recipient using IP Unicast. In case of a MEADcast to unicast transformation Layer 4 checksum correction is may required. Lastly, it is preferable, if MEADcast can be turned on and off, to ease the deployment of various level of network support.

Non-functional Given that the performance evaluation of MEADcast encompasses time-sensitive metrics such as latency and jitter, it is imperative that the clocks of each node are synchronized. This synchronization is crucial for ensuring comparable measurements. Additionally, similar routing procedures must be employed for IP Unicast, IP multicast, and MEADcast to maintain consistency in the evaluation. For instance, comparing software-based MEADcast routing with hardware-based IP routing would not yield meaningful results.

4. Implementation

This chapter focuses on the implementation of our experiment. Initially, Section 4.1 presents the implemented MEADcast specification. Subsequently, Section 4.2 delves into the technical details of the Kernel implementation of the router software. Section 4.3 provides insights into the sender implementation, including the interaction of discovery and data delivery phases and the receiver grouping algorithm. Finally, Section 4.4 demonstrates our designed network topologies, deployment of the testbed, and the conduction of the experiment.

4.1. Protocol Specification

Since the existing literature on MEADcast [TD18; DT19] lacks specific details regarding the header, such as specific field sizes or content (see Subsection 2.1.4), in this section we introduce the specifications utilized for our experiments.

Firstly, we omit the usage of an empty Hop-by-Hop IPv6 extension header as introduced by Tran and Danciu [TD18; DT19]. We do this for several reasons: First, to reduce the overhead of MEADcast. Second, if a non-MEADcast router attempts to process Hop-by-Hop extension headers the packet probably ends up in the router's slow path [CJ13], potentially harming the protocol's performance. Third, since the MEADcast header is intended to be processed exclusively by MEADcast routers, over which we maintain control, the Hop-by-Hop extension header serves no purpose. Lastly, according to various research studies, packets containing a Hop-by-Hop extension header experience an increased drop rate of up to 40% compared to packets without an extension header [Gon+16], especially across multiple Autonomous systems (ASs) [Gon+21; GL22].

Next, we introduce the detailed characteristics of the MEADcast header in our implementation. The first four octets of the MEADcast header constitute the static routing extension header, which is shared by all routing variants. This differs from previous implementations of MEADcast, which omitted the *Segments Left* field [Ngu19]. We decided to include this field to comply with RFC 8200 [DH17], reducing the likelihood that intermediate nodes drop MEADcast packets due to a malformed routing header. Since there is no existing *Routing Type* designated for MEADcast, we utilize the experimental values of 253 and 254 in accordance with RFC3692 [Nar04]. The *Segments Left* field remains fixed to zero and is never altered because, according to RFC 8200 [DH17], intermediate nodes that do not recognize the employed *Routing Type* value must ignore the routing header and proceed to process the next header. An illustration of the MEADcast header layout can be found in Figure 4.1, along with an in-depth description of each field in Table 4.1. Due to the time frame of this thesis, we omit the usage of the optional port list as it does not affect the core MEADcast processing.

Field	Description
Next Header	A 8-bit selector, identifying the type of the immediately following header [DH17]. Employs identical values as the IPv4 Protocol field (e.g. 17 = UDP, 59 = No Next Header for IPv6) [Aut24].
Hdr Ext Len	An 8-bit unsigned integer representing the length of the Routing header in 8-octet units, excluding the initial 8 octets [DH17].
Routing Type	An 8-bit identifier denoting a specific Routing header variant [DH17]. Given the absence of an existing Routing Type for MEADcast, we utilize the experimental values of 253 and 254 in accordance with RFC3692 [Nar04].
Segments Left	An 8-bit unsigned integer representing the number of remaining route segments, indicating the number of explicitly listed intermediate nodes yet to be traversed before reaching the final destination [DH17]. Remains fixed to zero and is not modified by MEADcast routers. Consequently, intermediate nodes that do not recognize the employed Routing type value are required to disregard the MEADcast header and proceed to process the subsequent header [DH17].
Num. Dst.	An 8-bit unsigned integer denoting the number of IPv6 addresses encoded in the address list. The field size sets a theoretical limit of 255 destinations.
Flags	A 2-bit Bitmap, valid combinations comprise `00`, `10`, and `11`.
Dcv	1 bit classifying the packet as a data (0) or discovery (1) packet.
Rsp	1 bit identifying the packet as a discovery request (0) or response (1) [DT19].
Hops	An 6-bit unsigned integer, specifying the distance between the sender and a router, denoted in the number of MEADcast hops. This field is utilized during the discovery phase. The sender initializes the field with zero, and it is incremented by each intermediate MEADcast router encountered. The field size sets a theoretical limit of 63 hops.
Reserved	A 16-bit reserved field. Should be initialized to zero during transmission and disregarded upon reception.
Delivery Map	A 32-bit Bitmap, where each bit at position i indicates whether a router at index i in the address list has already been delivered (0 = delivered, 1 = not yet delivered) [DT19].
Router Map	A 32-bit Bitmap, where each bit at position i indicates whether an address at index i in the address list is a router (0 = receiver, 1 = router) [DT19].
Address List	A variable-length list comprising the IPv6 addresses of receivers and MEADcast routers. The field size of the Delivery and Router Bitmap imposes a practical limit of 32 addresses.

Table 4.1.: Employed MEADcast header field description

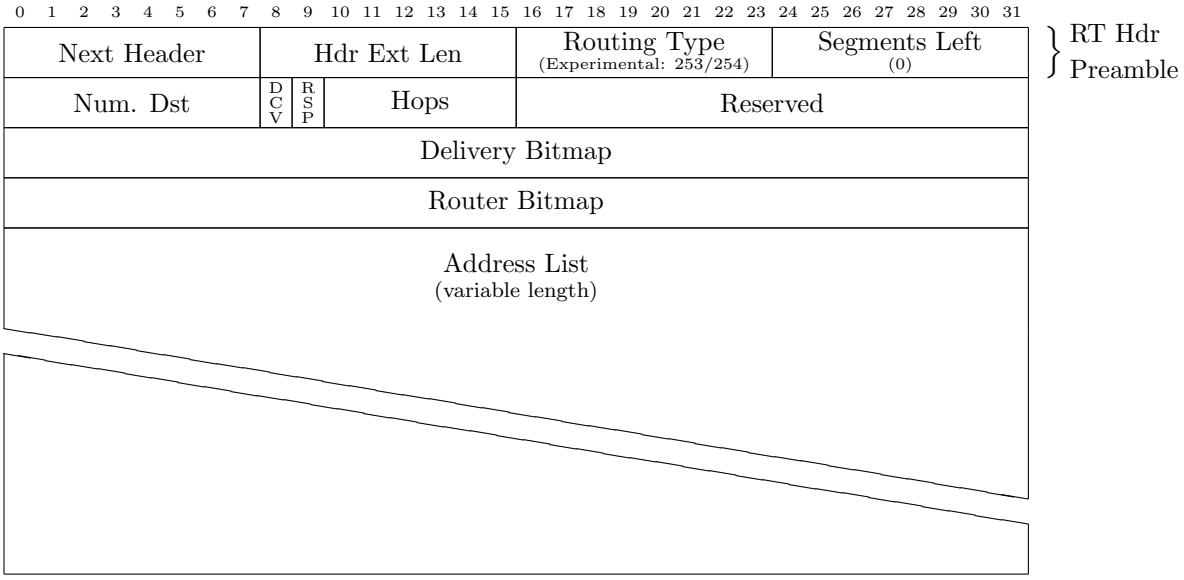

Figure 4.1.: Employed MEADcast header specification

4.2. Router

This section provides an overview of the MEADcast router implementation in the Linux Kernel. The MEADcast routing software is developed and tested using the latest stable Kernel release, version 6.5.3, at the time of development. MEADcast routing capabilities are tightly coupled to the flow of IPv6 packets through the Kernel network stack, making it infeasible to create a separate Kernel module loadable at runtime. Therefore, we decided to extend the IPv6 module with MEADcast capability. This feature can be enabled or disabled at compile-time using the `CONFIG_IPV6_MEADCAST` flag. To avoid the burden of recompiling or switching the kernel to enable or disable MEADcast support, we introduce a runtime configuration flag. This flag is managed through Sysctl, providing the virtual file `/proc/sys/net/ipv6/meadcast/enable`, which contains either 0 for disabled and 1 for enabled (see Listing A.9). For instance, the root user can enable MEADcast support by running the command `"echo 1 > /proc/sys/net/ipv6/meadcast/enable"`. However, it is crucial to ensure that this additional check does not adversely impact performance compared to a plain kernel, as it could potentially affect the reliability of experimental results.

Packets not designated to the router itself follow the "forwarding" packet flow in the Kernel. IPv6 extension headers of these packets are normally ignored by the Kernel (except for the Router Alert extension). Consequently, we embedded the entry point for MEADcast processing at the end of the `ip6_forward` method located in `net/ipv6/ipv6_output.c`. If a packet contains the routing header extension the `mdc_rcv` method located in `net/ipv6/mdcast.c` gets invoked. This method performs basic sanity checks on the MEADcast header and then invokes `mdc_dcv_rcv` for discovery and `mdc_data_rcv` for data packets.

Discovery: First, discovery packets get validated and the hop counter in the MEADcast header is incremented. To create a discovery response the socket buffer struct of the incoming discovery request is cloned. By this, we solely have to update the IP source and destination field of the cloned packet, set the response flag, and perform a routing table

lookup for the sender's address. The source IP address of the discovery response is set to the IP address associated with the interface returned from the routing lookup. Lastly, the `dst_output` method is invoked, dispatching the packet to the corresponding layer 2 handler. The processing of the original packet does not need further handling, as it is already correctly done by the default forwarding flow.

Data: Data packets undergo several steps upon arrival at the MEADcast router. Firstly, they are validated, and the hop counter in the MEADcast header is incremented. Subsequently, the router iterates over the Delivery and Router Bitmap to identify undelivered routers (indices set in both bitmaps). During this iteration, if the router is not the last undelivered router in the sequence, the socket buffer of the original packet is cloned and forwarded accordingly. However, if the router is the last undelivered router, the original packet can be forwarded without the need for costly replication. To determine whether to clone the packet, the routers monitors whether an undelivered router at index $r1$ has a successor (another undelivered router) at index $r2$, with $r1 < r2$. The algorithm's specifics are depicted in Figure 4.2.

For each undelivered router R_u specified in the MEADcast header, the processing router R_p decides whether to transmit the data via MEADcast or IP Unicast. If the IP address of R_u in the MEADcast header belongs to the processing router R_p itself, all designated endpoints will be delivered via IP unicast. Additionally, if the number of endpoints assigned to the considered router R_u from the MEADcast header is below a certain threshold, the processing router R_p performs a premature MEADcast to IP Unicast transformation. For example, if R_u has one designated endpoint, R_p could execute a premature MEADcast to IP Unicast transformation, potentially enhancing performance. The threshold can also be configured during runtime by setting the parameter `/proc/sys/net/ipv6/meadcast/min_dsts`. Setting `min_dsts` to zero disables premature MEADcast to IP Unicast transformation.

If the packet is forwarded via MEADcast, the router updates both the delivery bitmap and the destination field of the IPv6 header. In the delivery bitmap, all bits are set to zero except for the index corresponding to the considered router. Additionally, the IP destination field is set to the address of the first receiver following the router in the address list. Lastly, the router performs a routing table lookup for the new destination address and forwards the packet accordingly.

In case of IP Unicast transmission, the router converts the MEADcast packet into an IPv6 packet by copying the IPv6 header in front of the Layer 4 header and adjusting the socket buffer pointers accordingly. Additionally, in alignment with RFC 1624 [Rij94], the L4 checksum is incrementally updated due to the modified destination IP address.

4.3. Sender

This section provides an overview of our MEADcast sender implementation. Subsection 4.3.1 delves into the internal implementation specifics, such as threading and data structures. Subsection 4.3.2 provides an introduction into the algorithm which groups a list of receivers into MEADcast packets.

4.3.1. Software Architecture

To ensure efficient processing that is comparable to existing network protocol implementations, we chose to implement the sender in the C language. The main tasks of the sender

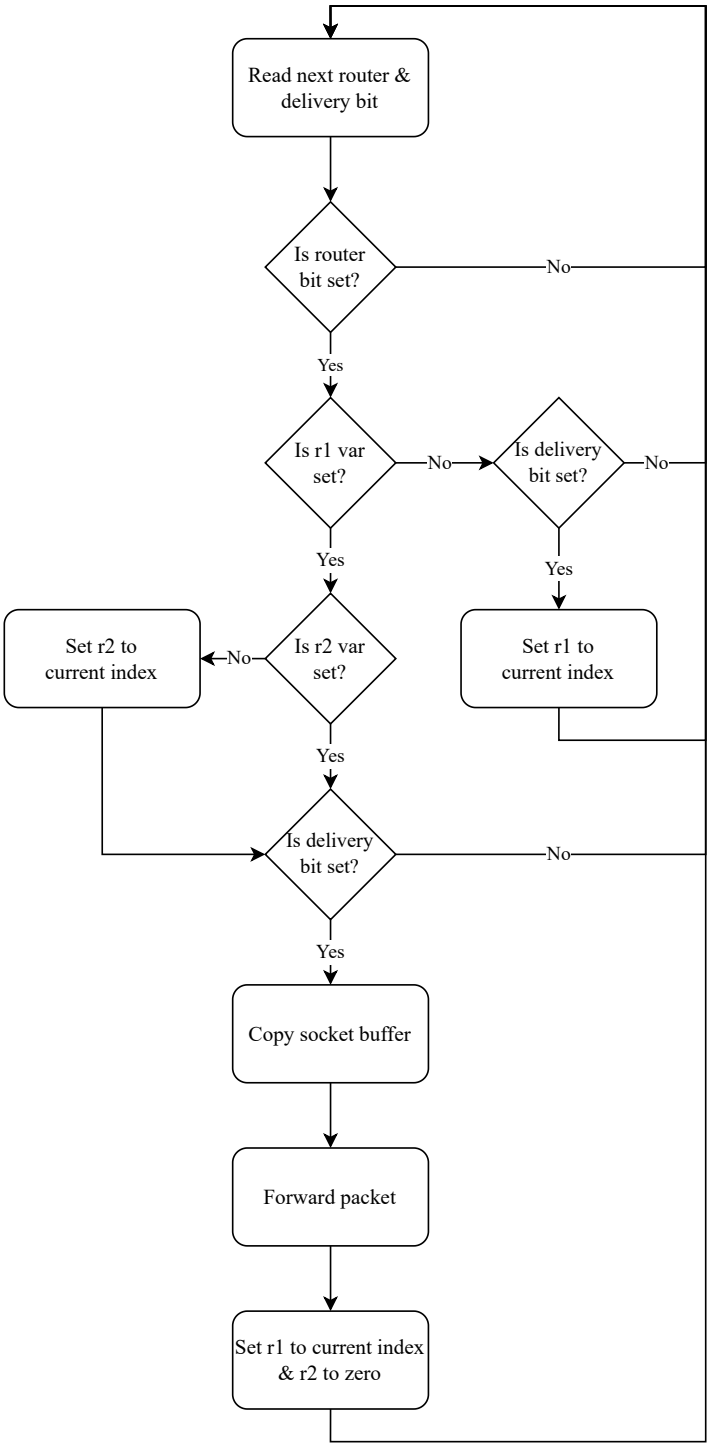

Figure 4.2.: Router: MEADcast header parsing

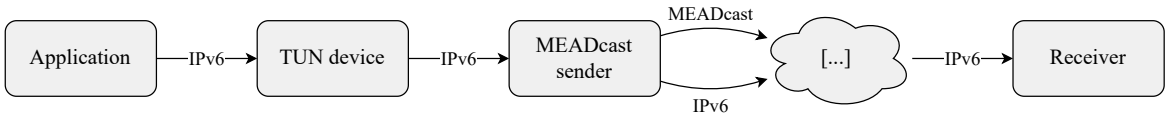

Figure 4.3.: Sender: TUN interface

include sending periodical discovery requests to all group members, receiving discovery responses, constructing a topology tree based on the received discovery responses, grouping receivers into MEADcast packets, and transmitting data to all group members either via MEADcast or IP Unicast.

Startup All receivers must be known at startup, dynamic runtime memberships are not supported. We regard this decision as justifiable, given that MEADcast does not define any mechanism for joining or leaving a group, and the primary objective of this thesis is to compare the protocol with existing alternatives. On startup the sender initializes a topology tree with itself as the root and all receivers as direct attached leaves. Moreover, a TUN interface is created representing the common interface to other applications. Any data, which is send to the TUN interface will be transmitted to the MEADcast group (see Figure 4.3. However, since MEADcast is a *1:n* multicast protocol, the sender does not handle traffic directed from the MEADcast receivers towards the sender. The Maximum transmission unit (MTU) of the TUN interface will be set to the MTU of the bind interface minus the maximum allowed MEADcast header size. This ensures, that all packets read from the TUN interface can be transmitted via MEADcast. Additionally, for convenience a host route to the TUN interface gets created automatically (default: `fd15::1`). Next, a discovery and transmission thread is created. This is necessary to facilitate simultaneous data transmission and periodic MEADcast discovery.

TX thread The transmission thread operates on a straightforward pattern. It reads data from the TUN device and transmits it to the MEADcast group either via MEADcast or IPv6 unicast. The key element for data transmission is the `tx_group` structure, which includes three members: `*mdc`, `nuni`, and `uni` (as detailed in Listing 4.1). These members encompass all the necessary information for data transmission, rendering `tx_group` as the only data structure required by the transmission thread. `*mdc` is a reference to a linked list of receivers grouped into MEADcast headers ready for transmission. `nuni` contains the count of unicast addresses stored in `uni[]`. Lastly, `uni[]` is an array of receivers served via IPv6 unicast.

The `tx_group` structure is the only data accessed by both the discovery and transmisson thread. Upon initialization of the transmission thread, a `tx_group` structure is instantiated with all receivers added to the `uni[]` list and an empty `*mdc` list. Subsequently, the `tx_group` is exclusively modified by the discovery thread. To prevent race conditions on the `tx_group`, the threads share an atomic reference pointing to the current group.

Next, the transmission thread enters an infinite loop executing the following steps. Initially, it blockingly reads data from the TUN device. Second, the sender atomically reads the current `tx_group`. This step is substantial because the `tx_group` can be updated by the discovery thread at any time. Finally, the data read from the TUN device is transmitted to the MEADcast group by first transmitting it via IPv6 unicast to all addresses listed in

```
struct tx_group {
    struct child *mdc;
    size_t nuni;
    struct addr uni[];
};
```

Listing 4.1.: Sender: tx_group structure

`uni[]` and subsequently transmitting it via MEADcast by copying the MEADcast headers stored in `*mdc` in front of the Layer 4 header. The copying of the MEADcast header overwrites the IPv6 header, necessitating the unicast transmission to occur before the MEADcast transmission.

Discovery Thread The discovery thread operates on a more intricate pattern compared to the transmission thread. Its primary tasks include sending periodic discovery requests to all group members, receiving discovery responses from intermediate MEADcast routers, maintaining the topology tree, and grouping receivers into MEADcast headers.

To manage the recurring discovery phase, the sender utilizes a discovery interval and a discovery timeout timer (see Figure 4.4). At the beginning of the discovery phase, the sender transmits discovery requests to all group members. The discovery timeout determines how long the sender waits for discovery responses, while the discovery interval determines the duration between two consecutive discovery phases. To monitor multiple timers, file descriptor-based timers are employed, which deliver timer expiration notifications via descriptors. This allows for efficient monitoring using `epoll` [Ker23b], an I/O event notification facility designed for monitoring multiple file descriptors [Ker23a].

On startup, the discovery thread initializes the timers and adds the file descriptors to `epoll`. Besides the timers, `epoll` also monitors the MEADcast file descriptor. Afterward, the discovery thread enters an infinite loop executing the following steps. Initially, the thread invokes `epoll_wait` (level-triggered), waiting for an event from one of the three file descriptors. It either waits for one of the timers to expire or a MEADcast packet to arrive.

On expiration of the interval timer, the discovery thread sends discovery requests to all group members, disarms the interval timer, and starts the discovery timeout timer.

Upon receiving a MEADcast packet, the sender performs basic sanity checks and validation. This includes verifying whether the address from the discovery response's address list is a MEADcast group member. To facilitate, a fast lookup of IPv6 addresses a Judy array is utilized. A Judy array is a sparse dynamic array with the key benefits of scalability, high performance, and memory efficiency [Bas04b]. More precisely, we are using the `JudyHS` functions, being a hybrid of the best features of hashing and Judy methods [Bas04a]. `JudyHS` outperforms a hashing method across smaller and larger populations than the optimal hash table size, without necessitating any tuning or configuration [Bas04a]. All receivers and routers are stored in the Judy array, using their IPv6 address as the hashing key. If a discovery response passes the checks and validation, the router is inserted into the topology tree or its existing entry is updated. The position in the tree is determined by the distance (hops) in the discovery response.

On expiration of the timeout timer, the sender starts grouping the receivers based on the

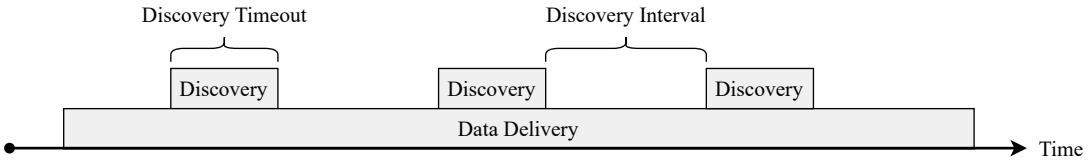

Figure 4.4.: Sender: Discovery Timers

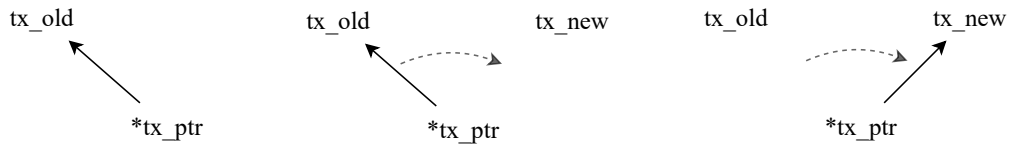

Figure 4.5.: Sender: Atomic update of the transmission group

topology tree. A new transmission group (`tx_new`) is created, and the atomic reference is updated to point to the newly created group (see: Figure 4.5). This procedure is required because complex structures can not be created atomically, prompting the decision to swing the atomic pointer after the new group's creation.

4.3.2. Grouping Algorithm

This section illustrates the receiver grouping algorithm, utilizing an exemplary topology tree depicted in Figure 4.6. Initially, Example 1 and Example 2 outline the procedural steps of the algorithm for fundamental configurations. Example 3, demonstrates the impact of the *"Merge Range"* parameter (see `"--merge"` in Listing A.3). Example 4, depicts the implication of the *"OK Address List Length"* parameter (see `"--ok"` in Listing A.3). Example 5 highlights the effect of the *"Minimum Leaf Count"* parameter (see `"--min-leaves"` in Listing A.3). A comprehensive description of all sender parameters is provided in Listing A.3. Furthermore, Algorithm 1 demonstrates the core grouping algorithm.

Example 1: Max. Address List Length: 5

In this example, we examine the topology depicted in Figure 4.6, with a maximum address list length of five. Initially, the sender S conducts a Depth First Search (DFS) to locate a router with unvisited leaves, yielding R_2 in our scenario. Subsequently, a new MEADcast header is generated, encompassing R_2 along with its unvisited leaves, which are then marked as visited. As the size of the address list is greater or equal to max -1, no further addresses can be added, resulting in a header of $\{R_2,E_1,E_2,E_3\}$. Another DFS is initiated to identify the next router with unvisited leaves, resulting in R_3. The procedure at R_3 mirrors that at R_2, resulting in $\{R_3,E_6,E_7,E_8\}$. The subsequent DFS yields R_1, and the procedure an identical procedure to that at R_2 is followed resulting in $\{R_1,E_4,E_5\}$. As no unvisited leaves remain, the final grouping is $\{R_2,E_1,E_2,E_3\}$, $\{R_3,E_6,E_7,E_8\}$, $\{R_2,E_1,E_2,E_3\}$.

Example 2: Max. Address List Length: 10

In this example, we analyze the topology depicted in Figure 4.6, with a maximum address list length of ten. The initial DFS returns R_2. Subsequently, a new MEADcast header is generated, encompassing R_2 along its unvisited leaves, which are then marked as visited. The current state of the header is $\{R_2,E_1,E_2,E_3\}$. If R_2 has a child router with unvisited leaves, we explore it. However, given that R_2 has no child router, we ascend to R_2's parent, R_1. As the address list of the current header has sufficient space left for R_1 and its unvisited leaves, they are appended, while the leaves are marked as visited. The current state of the header is $\{R_2,E_1,E_2,E_3,R_1,E_4,E_5\}$. If R_1 has a child router with unvisited leaves, we explore it, leading us to R_3. As the address list of the current header cannot accommodate R_3 and its leaves, a new packet is created, resulting in $\{R_3,E_6,E_7,E_8\}$. As no unvisited leaves remain, the final grouping is $\{R_2,E_1,E_2,E_3,R_1,E_4,E_5\}$, $\{R_3,E_6,E_7,E_8\}$.

Example 3: Max. Address List Length: 10, Merge Range: 1

In this instance, we examine the topology illustrated in Figure 4.6, with a maximum address list length of ten and a merge range of one. A merge range of one allows the sender S to to merge leaves from distinct parent routers under a common ancestor within a distance (MEADcast hops) of one. The initial DFS returns R_2. Subsequently, a new MEADcast header is generated, encompassing R_2 along its unvisited leaves, which are then marked as visited. The current state of the header is $\{R_2,E_1,E_2,E_3\}$. If R_2 has a child router with unvisited leaves, we explore it. However, given that R_2 has no child router, we ascend to R_2's parent, R_1. As the address list of the current header has sufficient space left for R_1's unvisited leaves, they are merged, while the leaves are marked as visited. The current state of the header is $\{R_2,E_1,E_2,E_3,E_4,E_5\}$. If the currently considered router (R_1) is closer to the sender S than the router in the address list (R_2), we set R_1 as the designated router, resulting in $\{R_1,E_1,E_2,E_3,E_4,E_5\}$. If R_1 has a child router with unvisited leaves, we explore it, leading us to R_3. As the address list of the current header has sufficient space left for R_3's unvisited leaves, they are merged, while the leaves are marked as visited. The current state of the header is $\{R_1,E_1,E_2,E_3,E_4,E_5,E_6,E_7,E_8\}$. Since the currently considered router (R_3) is not closer to the sender S than the router in the address list (R_1), there is no need to update the designated router in the address list. As no unvisited leaves remain, the final grouping is $\{R_1,E_1,E_2,E_3,E_4,E_5,E_6,E_7,E_8\}$.

Example 4: Max. Address List Length: 12, OK Addres List Length: 7

In this example, we examine the topology demonstrated in Figure 4.6, with a maximum address list length of twelve and a "OK" address list length of seven. A "OK" address list length of seven allows the sender S to prematurely finish a packet, if it contains an equal or greater number of addresses than seven. The initial DFS returns R_2. Subsequently, a new MEADcast header is generated, encompassing R_2 along its unvisited leaves, which are then marked as visited. The current state of the header is $\{R_2,E_1,E_2,E_3\}$. If R_2 has a child router with unvisited leaves, we explore it. However, given that R_2 has no child router, we ascend to R_2's parent, R_1. As the address list of the current header has sufficient space left for R_1 and its unvisited leaves, they are appended, while the leaves are marked as visited. The current state of the header is $\{R_2,E_1,E_2,E_3,R_1,E_4,E_5\}$. As the length of the address list is greater or equal to "OK" address list length, no further addresses are added to the list, even

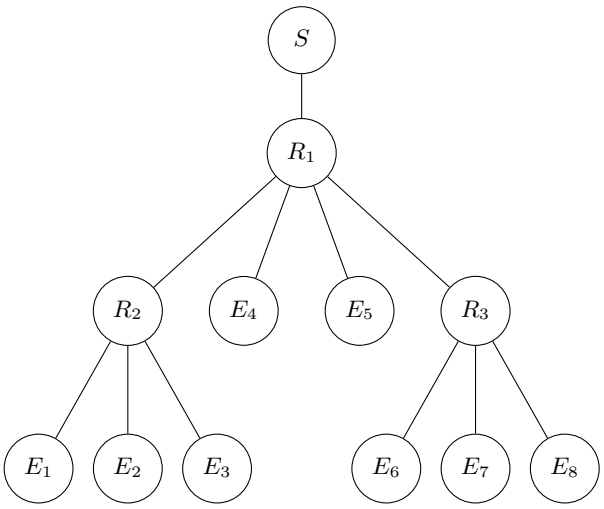

Figure 4.6.: MEADcast network topology tree. S represents the sender, R_i MEADcast routers, and E_j group members.

though R_3 and its unvistited leaves could be acommendated. Another DFS is initiated to identify the next router with unvisited leaves, resulting in R_3. The procedure at R_3 mirrors that at R_2, resulting in $\{R_3,E_6,E_7,E_8\}$. As no unvisited leaves remain, the final grouping is $\{R_2,E_1,E_2,E_3,R_1,E_4,E_5\}$, $\{R_3,E_6,E_7,E_8\}$.

Example 5: Max. Address List Length: 10, Min. Leaf Count: 3

In this instance, we examine the topology illustrated in Figure 4.6, with a maximum address list length of ten and a merge range of one. With a minimum leaf count of three the sender S marks all leaves of routers with less than three leaves for unicast delivery. The initial DFS returns R_2. Subsequently, a new MEADcast header is generated, encompassing R_2 along its unvisited leaves, which are then marked as visited. The current state of the header is $\{R_2,E_1,E_2,E_3\}$. If R_2 has a child router with unvisited leaves, we explore it. However, given that R_2 has no child router, we ascend to R_2's parent, R_1. As R_1 has less than three leaves, its leaves are appended to the unicast delivery list, and marked as delivered. If R_1 has a child router with unvisited leaves, we explore it, leading us to R_3. As the address list of the current header has sufficient space left for R_3's unvisited leaves, they are merged, while the leaves are marked as visited. The current state of the header is $\{R_2,E_1,E_2,E_3,R_3,E_6,E_7,E_8\}$. As no unvisited leaves remain, the final grouping is *MEADcast*: $\{\{R_2,E_1,E_2,E_3,R_3,E_6,E_7,E_8\}\}$, *Unicast*: $\{E_6,E_7\}$.

4.4. Experiment

This section illustrates the implementation of the experiment. Initially, Subsection 4.4.1 depicts the network topology and the selected designs for each use case. Subsequently, Subsection 4.4.2 delves into the deployment process of our virtual testbed. Finally, Subsection 4.4.3 provides insights into the conduction of each use case.

Algorithm 1 Sender: grouping algorithm

Input: Router (s) to start grouping at
Output: `tx_group` structure

```
1: group ← NULL                                          ▷ tx_group structure
2: m ← 0
3: n ← 0
4: r ← getStart(s)                  ▷ Gets next router based on depth-first search
5: while r do
6: start:
7:     while r.freeLeaves > 0 do
8:         m ← n + r.freeLeaves + 1
9:         if m > max then
10:            go to start
11:        else if m ≥ max − 1 then
12:            addRouterToGroup(r)                  ▷ Update bitmaps and address list
13:            recursiveBackpropergate(r)  ▷ Update leaf and child count of all anchestors
14:            go to next
15:        else if m < max then
16:            addRouterToGroup(r)
17:            break
18:        end if
19:    end while
20:    while r.freeChild > 0 do
21:        c ← getFreeRouter(r)
22:        if c then
23:            r ← getRouter(c)
24:            go to start
25:        end if
26:    end while
27:    if r.node.parent then
28:        backPropagate(r)                  ▷ Update parent leaf and child count
29:        r ← getRouter(r.node.parent)
30:        if not r.node.parent then                ▷ If parent is root finish group
31:            go to next
32:        end if
33:        go to start
34:    end if
35: next:
36:    finishGroup()                ▷ Adds MEADcast header to group.mdc list
37:    r ← getStart(r)
38: end while
39: addRemainingLeaves(s)          ▷ Adds the remaining leaves to group.uni list
40: return group
```

4.4.1. Network topology

This section provides detailed insights into the designed experiment topology. Furthermore, it highlights the specific selection of sender, receivers, and routers for each use case.

Topology

The architecture of our testbed aligns with the in Subsection 3.6.1 formulated requirements, achieved through the implementation of the hierarchical three-layer model.

The testbed comprises 205 nodes consisting of 40 routers and 165 end devices. The network is divided into four domains, each potentially representing distinct buildings on a university campus. All four domains encompass seven to nine client networks, accommodating five end devices each. The access layer of each domain comprises up to seven routers (3-digit router IDs in Figure 4.7). Since this thesis evaluates a transport protocol, we waive the application of switches on the access layer, deploying solely layer 3 devices. The distribution layer of each domain encompasses two to three interconnected routers (2-digit router IDs), aggregating the access layers within its respective domain and facilitating the connection to the core layer. The testbed's core consists of routers routers (1-digit router IDs) interconnecting the four network domains. Additionally, the backbone possesses a high connectivity, ensuring both high availability and redundancy. To reduce management and operational overhead, the testbed incorporates fewer redundant links from the access to the distribution layer and from the distribution to the core layer compared to the recommendations provided by Cisco and Huawei [Cis14; Hua23]. This trade-off is considered acceptable, as this thesis does not delve into the details of the availability and fault tolerance of the network architecture. Additionally, router and link failures are artificially induced in any case.

The chosen architecture represents a realistic medium-sized network topology. The application of the hierarchical three-layer model ensures the required flexibility of the testbed. Various placements of senders, receivers, and routers are possible, allowing experiments with a receiver distribution ranging from high clustering within a single domain to a uniform spread across all four network domains. Moreover, the availability of alternative routes facilitates the representation of dynamic network environments, encompassing scenarios such as deliberate routing changes or router and link outages.

UC1: Live Stream

In accordance with the requirements outlined for UC1 in Paragraph 3.4, Figure 4.8 illustrates the chosen topology, which comprises up to 96 nodes, consisting of a maximum of 70 receivers (with a maximum of five per client network) and 26 routers. The configuration encompasses four network domains each representing a building on a campus or even an entire location. Client C_{3001} is designated as the sender. Areas within client networks C_{21}-C_{23}, C_{35}-C_{36}, and C_{41}-C_{46} exhibit high receiver clustering, while networks C_{11}, C_{14}, C_{16}, C_{28}, and C_{38} demonstrate low receiver density. The selection of client networks encompasses varying distances between sender C_{3001} and receivers, ranging from four hops for C_{35} and C_{36} to eight hops for C_{28}.

The chosen topology aligns with the requirements formulated in Paragraph 3.4 and facilitates the measurement of MEADcast's performance for mid to large-sized groups, incorporating a mixture of highly clustered and isolated nodes. This highlights the protocol's potential in scenarios of heterogeneous receiver distributions.

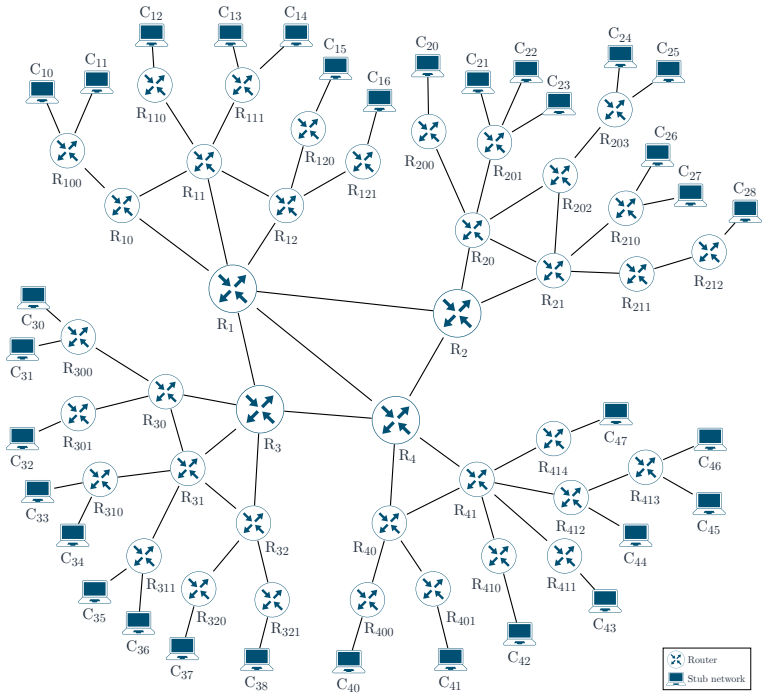

Figure 4.7.: Testbed Topology

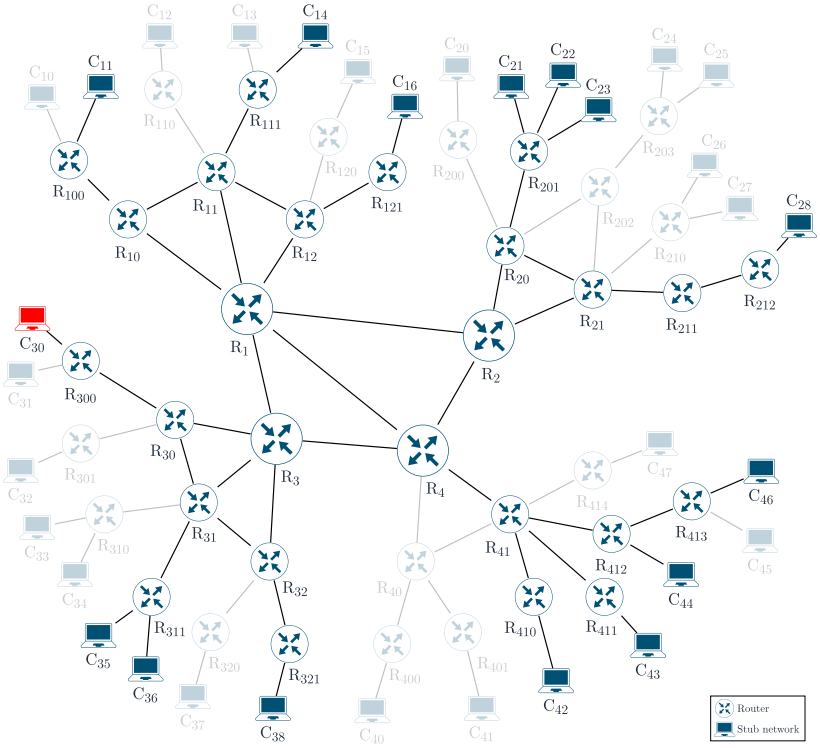

Figure 4.8.: UC1: Live Stream (C_{3001} *is sender*)

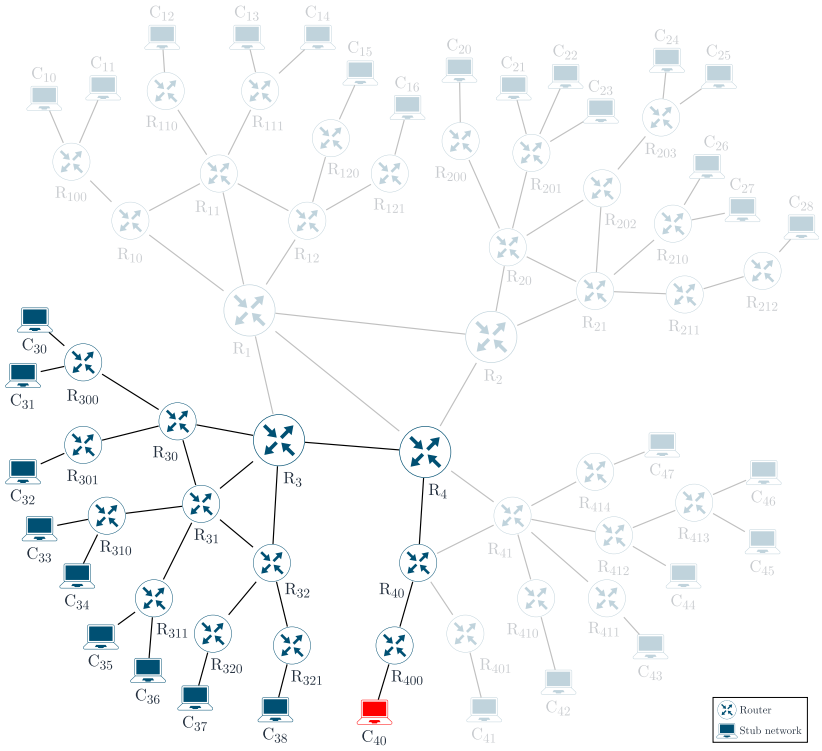

Figure 4.9.: UC2: File Transfer (C_{4001} *is sender*)

UC2: File transfer

In accordance with the requirements provided for UC2 in Paragraph 3.4, Figure 4.9 illustrates the chosen topology, consisting of up to 58 nodes, comprising 45 receivers (with a maximum of five per client network) and 13 routers. The node selection encompasses the entire network domain three, representing either a server network or even an entire data center, with the sender C_{4001} serving as the package source mirror. All client networks (C_{30}-C_{38}) are located within one highly clustered area. Furthermore, we have the flexibility to adjust the number of receivers by scaling the number of active nodes within a client network within a range of one to five.

The chosen topology aligns with the requirements outlined in Paragraph 3.4 and facilitates the evaluation of MEADcast's performance for mid to large-sized groups of highly clustered nodes, demonstrating the protocol's potential in scenarios with ideal conditions.

UC3: P2P Video conference

In accordance with the requirements formulated for UC3 in Paragraph 3.4, Figure 4.10 depicts the chosen topology, including a total of up to 78 nodes, comprising 60 clients (with a maximum of five per client network) and 18 routers. However, since this use case aims to assess MEADcast's suitability for small to medium-sized P2P conferences, the number of clients is typically lower. The topology features local clustering in the area of C_{33}-C_{36} and C_{21}-C_{25}, representing sub-teams in different office locations. Conversely, C_{12}, C_{40}, and C_{46} exhibit a low receiver density representing customers or service providers in remote areas.

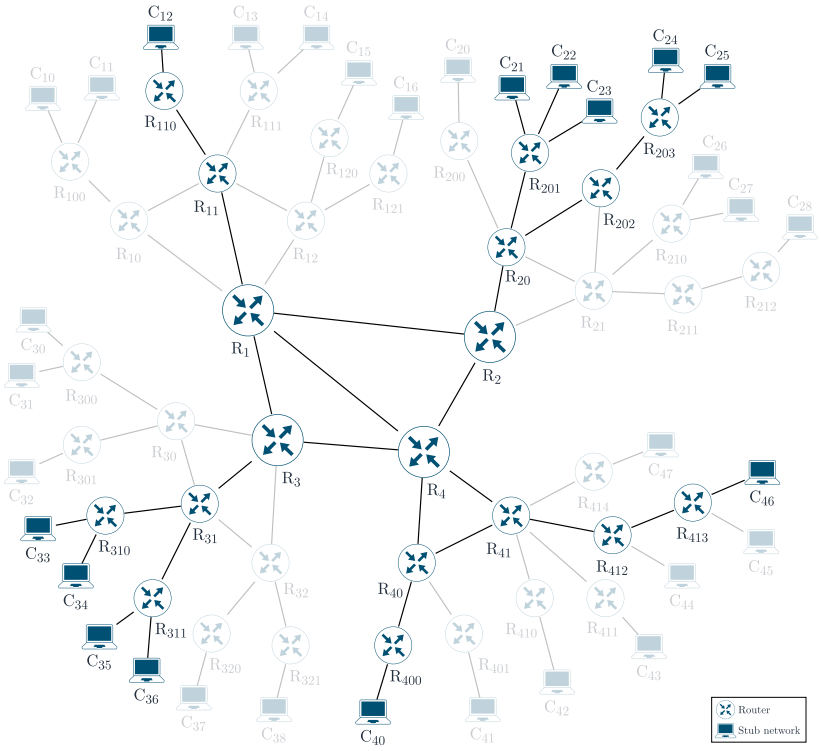

Figure 4.10.: UC3: P2P Video Conference (*all members are senders*)

The selection of client networks encompasses varying distances between clients, ranging from one hop within local clusters such as between C_{33} and C_{34}, up to eight hops between C_{25} and C_{46}. Since this use case focuses on P2P communication, all clients transmit and receive traffic.

The chosen topology aligns with the requirements stated in Paragraph 3.4 and facilitates the investigation of MEADcast's suitability for P2P communication and whether the protocol can bridge the asymmetric access link bandwidth. Moreover, the topology enables the examination of the effect of high MEADcast traffic volume on MEADcast routers.

UC4: Online Gaming

In accordance with the requirements outlined for UC4 in Paragraph 3.4, Figure 4.11 illustrates the chosen topology, comprising up to 72 nodes, including 45 receivers (with a maximum of five per client network) and 27 routers. However, since this use case aims to assess MEADcast's suitability for small to medium-sized groups, the number of clients is typically lower. The design features an evenly spread distribution of client networks with a low receiver density. No router is directly connected to more than one client network. Client C_{4501} is designated as the sender. The selection of client networks encompasses varying distances between sender and receivers, ranging from four hops for C_{43} up to eight hops for C_{24} and C_{28}.

The chosen topology design aligns with the requirements provided in Paragraph 3.4 and facilitates the evaluation of MEADcast's implication on latency and jitter. Moreover, the

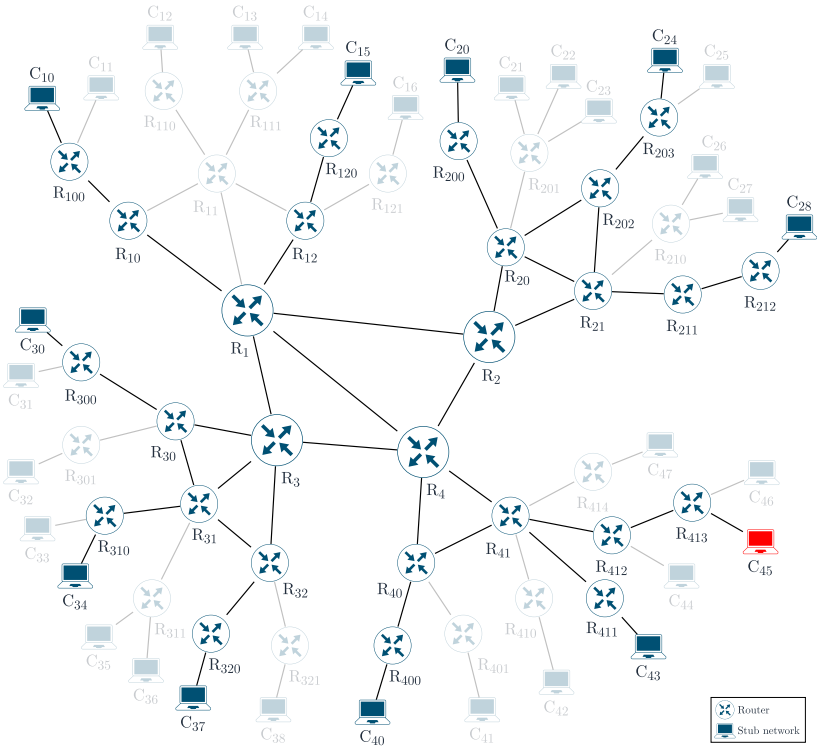

Figure 4.11.: UC4: Online Gaming (C_{45} *is sender*)

topology enables the investigation of MEADcast's performance in scenarios with suboptimal conditions, particularly with a high receiver distribution.

4.4.2. Technical Infrastructure

Given the impracticality of creating a physical testbed, we have opted for a virtual environment using Kernel-based Virtual Machine (KVM). The testbed comprises Virtual Machines (VMs) connected by virtual networks. The hypervisor, based on Debian 11.3, is equipped with 4 CPUs, 8 threads, 32 GB memory, and 120 GB of storage capacity.

Each VM is configured with 1 vCPU, 256 MB, and 8 GB of storage. To optimize resource utilization, we employ the para-virtualized "virtio" network adapter for the VMs. We utilize two types of VMs: clients and routers, both based on a Debian 11 no-cloud image. No-cloud images streamline the setup process by bypassing the need for OS installation. Additionally, direct Kernel boot is facilitated for the router, enabling swift kernel exchange by simply replacing the Kernel image file on the hypervisor.

During the build process, a router and a client template image are created. These serve as *"Qemu"* backing files for the VM images, ensuring that VMs only have to monitor changes compared to the backing template, reducing storage usage and build time significantly.

Automating the build process is a Python script provided in the digital appendix, with the command to construct our topology shown in Listing A.1. The script presents a wrapper around tools such as "*virsh*", "*libguestfs-tools*", and "*qemu-img*". It accepts several input parameters, including template configuration files for the virtual network generation,

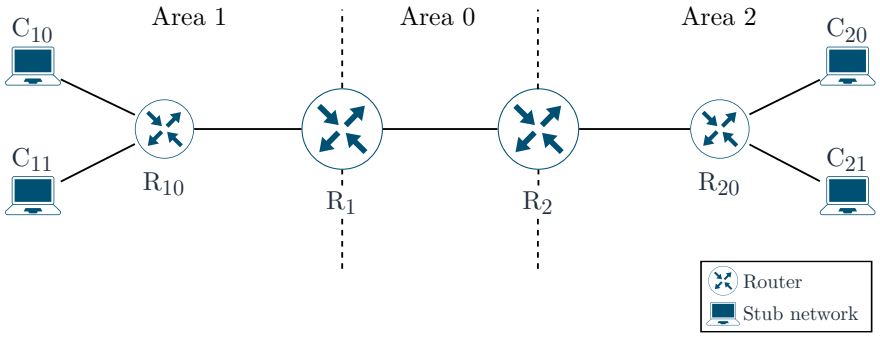

Figure 4.12.: Example topology generated from the configuration file shown in Listing A.2.

"*chrony*" (time synchronization), and "*frr*" (routing). Additionally, the script requires a Kernel image for the router and a configuration file describing the network topology. Listing A.2 demonstrates an exemplary topology definition, resulting in the topology shown in Figure 4.12. The `"stub"` key from the topology configuration contains a list of objects describing a stub network, its number of clients, the upstream router, and the OSPF area. The `"trans"` key includes a list of objects comprising a tuple of two router IDs and OSPF area. The routers are interconnected according to this list. The `"area_boundary"` key consists of a list of objects representing the OSPF boundaries, by router ID, OSPF area ID, and the network mask of the published summarizing route.

The IP naming scheme for client machines follows the format `"fd14::<AREA ID>:<ROUTER ID>:<CLIENT ID>"`. Additionally, each VM is attached to an IPv4 management network. The clients utilize static routing, with a default IPv4 route on the management interface and a "fd14::/16" route on the upstream interface.

For accurate measurement of latency and jitter, synchronized clocks among the VMs are imperative. While Network Time Protocol (NTP) is a common solution for time synchronization, Linux provides an alternative mechanism that does not require an NTP server or the network stack at all. Instead, the host injects para-virtualized Precision Time Protocol (PTP) devices into the guests facilitating highly accurate time synchronization. Apart from loading the `"kvm_ptp"` Kernel module and configuring *"chrony"* as usual, setting the clock source on the host via Kernel command line parameters to `"clocksource=tsc"` was necessary.

The router operates on our custom MEADcast Kernel version 6.5.2. *"Frr"* facilitates the routing stack, which supports a variety of routing protocols such as BGP, OSPF, RIP, and PIM [Pro]. OSPF is utilized as the IP Unicast routing protocol and PIM for IP Multicast routing. Beyond the standard setup of *"frr"*, manual alteration of the package manager source to `https://deb.frrouting.org/` was necessary, as the package from the default source misses the IPv6 PIM daemon. It is noteworthy while OSPF mandates routers to possess only a link-local address, the MEADcast discovery phase necessitates each router to have at least one global IPv6 address.

		Experiment			MEADcast					
		Clients				Discovery			Grouping	
Use case	Proto.	Tot.	per Router	Vol. *(MB/s)*	Net. Supp.	Dly *(s)*	Intvl *(s)*	T. out *(s)*	Max. size	Merge range
Stream	Uni	14	1	1	-	-	-	-	-	-
Stream	Multi	14	1	1	-	-	-	-	-	-
Stream	Mead	14	1	1	100%	0	0	0	10	0
Stream	Mead	14	1	1	100%	0	5	2	10	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Stream	Mead	14	1	1	100%	0	0	0	10	2
Stream	Mead	14	1	1	L1	0	0	0	20	0
Stream	Uni	28	2	2.5	-	-	-	-	-	-
Stream	Multi	28	2	2.5	-	-	-	-	-	-
Stream	Mead	28	2	2.5	100%	0	0	0	10	0
Stream	Mead	28	2	2.5	100%	0	5	2	10	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Stream	Mead	28	2	2.5	100%	0	0	0	10	2
Stream	Mead	28	2	2.5	L1	0	0	0	20	0

Table 4.2.: Use case, experiments and measurements. Excerpt of two experiments comprising multiple measurements conducted within UC1: Live Stream.

4.4.3. Conduction

We conducted over 180 measurements across three distinct use cases (UC1-3), repeating each measurement multiple times to ensure the validity and reliability of the collected data. Specifically, we conducted more than 120 measurements for UC1, 50 for UC2, and 10 for UC3.

Each *use case* is delineated by its compilation of routers, transit and stub networks, and specific application characteristics, including communication and traffic pattern. An *experiment* entails executing a series of measurements for a given use case, featuring a specific number of clients and data volume. This series comprises measurements for IP Unicast, IP Multicast, and various MEADcast configurations. A MEADcast configuration is composed of the sender parameterization and the network support level. For instance, Table 4.2 provides an excerpt of two experiments conducted within UC1: Live Stream.

Measurment We utilized *"Iperf2"* for the data transmission and to collect performance metrics including sender and receiver bandwidth utilization, latency, jitter, and packet loss. The usage of *"Iperf3"* was not feasible due to the necessity of an initial Transmission Control Protocol (TCP) handshake.

To measure the total network bandwidth utilization, we monitored all links within the use case. As the experiments were conducted in a virtual environment, we simply needed to measure the receive and transmission bandwidth of each virtual network, which is essentially

a network bridge on the host system. For this purpose, we utilized *"ifstat"*. To measure CPU and memory utilization, we employed a tool called *"sar"*. However, during the majority of our measurements, resource utilization peaked, hindering our ability to obtain meaningful results.

RX/TX To conduct IP Unicast measurements for UC1 and UC2 we connected to all receivers using GNU parallel and SSH, initiated Iperf in server mode and detached from the process (see Listing A.6). On the sender we initiated one instance of *"Iperf"* per receiver using GNU parallel.

For the IP Multicast measurements, the procedure on the receiver side is identical to IP Unicast, except that Iperf listens on a IPv6 Multicast address (see Listing A.7). On the sender side, one instance of Iperf is started to transmit traffic towards all multicast listeners.

To measure MEADcast, the procedure on the receiver side is identical to IP Unicast (see Listing A.8). On the sender we started the MEADcast sender implementation and configured Iperf to send its traffic into the TUN device, as shown in Figure 4.3. It is noteworthy, that we initiated the Iperf client (our sender) with the `"--no-udp-fin"` flag to prevent the Iperf server (our receivers) from sending a final report to the client, potentially falsifying our bandwidth measurements.

To conduct IP Unicast measurements for UC3 (P2P), we employed one instance of Iperf in server mode on each client (n), along with $n-1$ instances of Iperf in client mode, sending traffic to each other. P2P Multicast measurements were not feasible because Iperf only supports SSM. The procedure for P2P MEADcast measurements is similar to the 1:n measurement, except that the MEADcast sender is initiated on each client.

MEADcast configuration In each experiment, we conducted measurements for a variety of MEADcast configurations.

To gain insides into both the performance of the data delivery phase itself and the implications of the discovery phase, we measured MEADcast's performance with and without considering the discovery phase. To measure MEADcast's performance without discovery phase, we started the sender with a discovery interval of zero, resulting in the sender performing a single discovery phase during startup. After the discovery phase was completed, we conducted our measurement solely collecting data from the data delivery phase. For measuring the performance of MEADcast with discovery phase, we initialized the sender with the wait flag set, a delay of zero, a discovery timeout of 2 seconds, and a discovery interval of 5 seconds. By setting the wait flag, the sender delays the initial discovery until it receives traffic from the TUN device.

Furthermore, we conducted measurements with varying levels of network support, encompassing scenarios such as 100% support across all network layers (distribution, access, and core), support solely on the distribution (L2) and access layers (L3), support exclusively on the core layer (L1), and MEADcast support limited to specific routers.

To explore the performance implications of receiver grouping, we conducted measurements with various sender configurations. These configurations involved different maximum numbers of addresses per packet, ranging from 6, 10, 15, and 20, up to 32. Additionally, the `"--ok"` parameter was consistently set to 80% of the maximum.

UC1: Live Stream Initially, we conducted experiments for UC1 with a data volume of 5 Mbit/s, a common requirement for HD video streams [Car21]. However, during the measurements, it became evident that our hardware resources were insufficient to support this data volume as the group size increased. Consequently, we adjusted our experiments to use a baseline data volume of 1 Mbit/s (for audio stream) and 2.5 Mbit/s (for SD video stream) [Car21]. Additionally, we varied the number of receivers per router from 1 to 5. As a result, we conducted a total of 10 experiments comprising 14 to 70 receivers, with data volumes of 1 and 2.5 Mbit/s. However, even the experiment with 70 clients and a data volume of 2.5 Mbit/s exceeded our resource capacities, resulting in unreliable data.

UC2: File Transfer To simulate recurring bursts of high traffic volume, we needed to maximize the utilization of available bandwidth. Since *"Iperf"* lacks a feature to send as much data as possible for UDP, we increased the bandwidth parameter until we observed a packet loss of approximately 1.5%. Initially, we conducted experiments with a data volume of 25 MB and 100 MB. However, as shown in Table B.1, the relative savings for 25 MB and 100 MB were identical. Therefore we discontinued the 100 MB experiments. Subsequently, we conducted experiments with 1, 3, and 5 clients per router (totaling 9, 27, 45), with a data volume of 25 MB.

UC3: P2P Video Conference Initially, we conducted an experiment with twelve clients (one per stub network) and a data volume of 1 MB/s, a requirement commonly seen in video conferences [Car21]. However, even in this modest experiment, we observed packet loss rates of 50%. Due to the unreliable data, resulting from our resource limitations we decided to discontinue experiments for UC3.

UC4: Online Gaming The primary objective of UC4 was to evaluate the potential impact of MEADcast processing on latency and jitter. However, due to the constrained time frame of this thesis and the fact that latency and jitter metrics were collected throughout all previous measurements without any discernible impact on jitter, we opted to exclude this use case from our study.

Dynamic network environments Dynamic network environments were explored to asses the impact of common phenomena such as routing changes, network disruptions, and packet dropping by intermediate nodes on MEADcast. The measurements were conducted within the topology of UC3, with C_{1201} acting as the sender and three receivers located in the stub networks C_{33}-C_{36} each. These effects were consequently induced on the link between R_1 and R_3.

A route change was simulated by altering the link state on R_3's *"enp2s0"* interface from "up" to "down". This action prompted the underlying routing protocol on R_1 to immediately switch to an alternative route via R_4.

To simulate a link failure, a firewall rule using *"Iptables"* was applied on R_3 to drop any traffic received on the link towards R_1. The corresponding commands are detailed in Listing A.4. Similar to the route alteration experiment, R_1 adapted to an alternative route via R_4.

For simulating packet dropping by an intermediate node, a firewall rule was employed on R_3 to drop packets containing an IPv6 routing header extension on the link towards

R_1. First, MEADcast transmission with periodic discovery phase was initiated, followed by enabling the firewall rule during MEADcast data transmission. The associated commands are provided in Listing A.5. After eight seconds this rule was disabled again, to observe MEADcast reverting to its initial grouping. This experiment was conducted with both 100% network support and support only on L2 and L3 in network domain 3 (R_{31}, R_{310}, R_{311}). With 100% network support, the implications of MEADcast falling back to another router located in front of the firewall were measured. Network support only on L2 and L3 in network domain 3, enabled the observation of MEADcast falling back to IP Unicast transmission.

Anomaly handling MEADcast's anomaly handling was investigated by intentionally triggering anomalies and observing the system's response. This involved sending manually crafted discovery responses to the sender using the *"Scappy"* tool. Furthermore, we simulated a router's failure to perform MEADcast processing by disabeling MEADcast during the data transmission phase.

5. Evaluation

Following the implementation (see Chapter 4) and conduction of our experiment, Section 5.1 presents the results obtained from our measurements. Subsequently, Section 5.2 discusses the implication of our findings, aiming to address the research questions formulated in Section 3.3.

5.1. Results

This section presents the results of our experiments. Subsection 5.1.1 provides a comparison of performance metrics among IP Unicast, MEADcast, and IP Multicast, encompassing total network bandwidth utilization, sender upstream bandwidth utilization, and total transfer time. Subsection 5.1.2 delves into the implications of different sender configurations and varying levels of MEADcast network support on these performance metrics, as well as link bandwidth utilization, and latency. In Subsection 5.1.3, we examine the effect of the discovery phase on metrics such as bandwidth utilization, latency, and packet loss. Lastly, Subsection 5.1.4 depicts the impact of events commonly observed in dynamic network environments, such as route change, link outages, and intermediate nodes dropping packets on MEADcast delivery.

5.1.1. Performance Metrics

The conducted experiments consistently demonstrated that MEADcast achieved better results in all metrics compared to Unicast. Across all MEADcast configurations tested, there was a significant reduction in total network bandwidth utilization, sender upstream bandwidth utilization, and transmission time.

Total Network Bandwidth On average, the total network bandwidth was reduced by 50.29% in UC1: Live Stream (1 MB/s for 45s), 58.78% in UC1: Live Stream (2.5 MB/s for 45s), and 58.93% in UC2: File Transfer (25 MB) compared to IP Unicast (see Table 5.1). Moreover, as the number of clients per router increased from one to five, the savings for UC1 and UC2 rose in total by 25% and 30% respectively. The relative reduction remained constant across varying data volumes. In comparison, IP Multicast achieved an average reduction of 80% in network bandwidth compared to IP Unicast. As the number of clients per router increased from one to five, the savings rose in total by 30%, from 60% for one client per router up to over 90% for five clients per router.

Sender Upstream Bandwidth Furthermore, we observed a significant decrease of the sender upstream bandwidth consumption compared to IP Unicast. On average, it was reduced by 78.06% in UC1 (1 MB/s), 81.03% in UC1 (2.5 MB/s), and 84.61% in UC2 (25 MB) (see Table 5.2). In comparison, IP Multicast achieved relative reductions of 96.71%, 97.33%, and 94.41%, respectively. Similar to the findings for the total network bandwidth, the relative

Clients *(per Router)*	**UC1** *(1 MB/s)*		**UC1** *(2.5 MB/s)*		**UC2** *(25 MB)*	
	Mead	Multi	Mead	Multi	Mead	Multi
1	25.50%	59.59%	-	-	42.11%	64.69%
2	49.20%	79.79%	50.13%	79.76%	-	-
3	56.43%	86.53%	53.17%	84.29%	63.30%	88.24%
4	60.28%	89.89%	60.34%	86.35%	-	-
5	60.06%	91.45%	71.49%	88.62%	71.38%	94.24%
Average	50.29%	81.45%	58.78%	84.76%	58.93%	82.39%

Table 5.1.: Total network bandwidth reduction *(relative to unicast)*

reduction remained constant across various data volumes. As the number of clients per router increased from one to five, the savings for UC1 (1 MB/s) rose by 15% from 64.61% for one client per router up to 80.95% for five clients.

Clients *(per Router)*	**UC1** *(1 MB/s)*		**UC1** *(2.5 MB/s)*		**UC2** *(25 MB)*	
	Mead	Multi	Mead	Multi	Mead	Multi
1	64.61%	92.90%	-	-	79.84%	89.03%
2	78.12%	96.47%	78.08%	96.45%	-	-
3	81.97%	97.67%	80.81%	97.37%	87.65%	96.36%
4	84.65%	98.22%	84.77%	97.86%	-	-
5	80.95%	98.31%	80.45%	97.64%	86.33%	97.86%
Average	78.06%	96.71%	81.03%	97.33%	84.61%	94.41%

Table 5.2.: Total sender upstream bandwidth reduction *(relative to unicast)*

Transfer Time Our experiments revealed significant reductions in total transfer time with MEADcast. On average, MEADcast reduced the total transfer time by 8.94% in UC1 (1 MB/s), 8.93% in UC1 (2.5 MB/s), and 24.16% in UC2 (25 MB). This emphasizes that, given MEADcast's lower bandwidth consumption, transfer time can be significantly reduced in scenarios where network or sender upstream bandwidth poses a bottleneck.

However, it is crucial to acknowledge that the observed time savings in UC1, which is fixed to 45 seconds of data transmission, may be influenced by CPU limitations. As the unicast transmission requires, one *"Iperf"* instance per receiver, scheduling significantly more *"Iperf"* processes than available CPUs is necessary for high numbers of receivers. The fact, that notable time savings in UC1 only occur for a high number of clients supports this hypothesis. In contrast, UC2 consistently exhibits reduced transfer times across all numbers of clients. Consequently, our subsequent discussion will focus on the results from UC2.

Figure 5.1 illustrates that MEADcast possesses a significantly lower growth rate in transfer time, as the number of group members increases compared to IP Unicast. It scales more similarly to IP Multicast in our experiments. For instance, with 5 clients per router, Multicast reduced the transfer time by 77.97%, from 2 minutes and 46 seconds to 36 seconds,

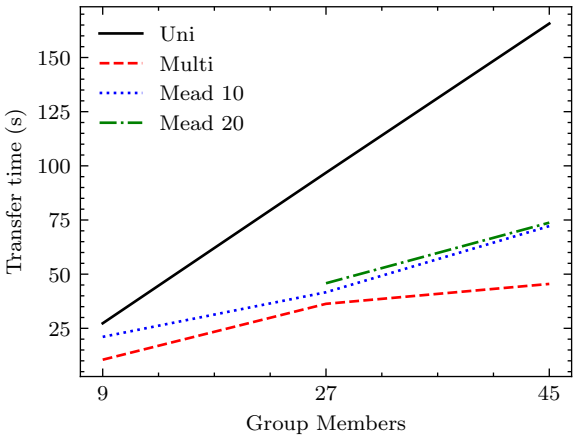

Figure 5.1.: UC2: File Transfer Comparison of total transfer time for a data volume of 25MB.

while MEADcast achieved a reduction of 57.46%, resulting in a transfer time of 1 minute and 11 seconds (see Table 5.3).

Clients *(per Router)*	**UC1** *(1 MB/s)*		**UC1** *(2.5 MB/s)*		**UC2** *(25 MB)*	
	Mead	Multi	Mead	Multi	Mead	Multi
1	0.65%	0.66%	-	-	37.94%	61.54%
2	1.11%	1.10%	2.93%	2.90%	-	-
3	2.27%	2.27%	18.43%	18.55%	52.14%	62.50%
4	10.39%	10.40%	36.19%	36.24%	-	-
5	30.30%	30.22%	38.82%	38.94%	57.46%	77.97%
Average	8.94%	8.93%	24.09%	24.16%	49.18%	67.34%

Table 5.3.: Total transfer time reduction *(relative to unicast)* The reduction observed in UC1 is primarily attributed to CPU limitations. This is because, in IP Unicast measurements with a high number of clients, the sender schedules significantly more *"Iperf"* processes than available cores.

Figure 5.2, illustrates the performance enhancements in two specific scenarios, with UC1 on the left-hand side and UC2 on the right. In both cases, MEADcast substantially reduces network bandwidth utilization, achieving reductions of over half compared to IP Unicast (UC1: 64%, UC2: 71%) (see Figure 5.2 (a) and (c)). Moreover, MEADcast achieves even greater reductions in sender upstream bandwidth utilization, reaching 87% in UC1 and 92% in UC2 compared to IP unicast (see Figure 5.2 (b) and (d)). At the start of MEADcast transmission, there is a discernible impact from the initial discovery phase. During this phase, both network and upstream bandwidth experience peaks for approximately 2 seconds, reaching similar values as IP Unicast, and even surpassing them in UC2. The bandwidth utilization in UC2 fluctuates more compared to UC1 because it pushes the testbed's resource usage to its limits. In UC2, MEADcast significantly reduces transmission time by 55% compared to IP Unicast (see Figure 5.2 (c) and (d)).

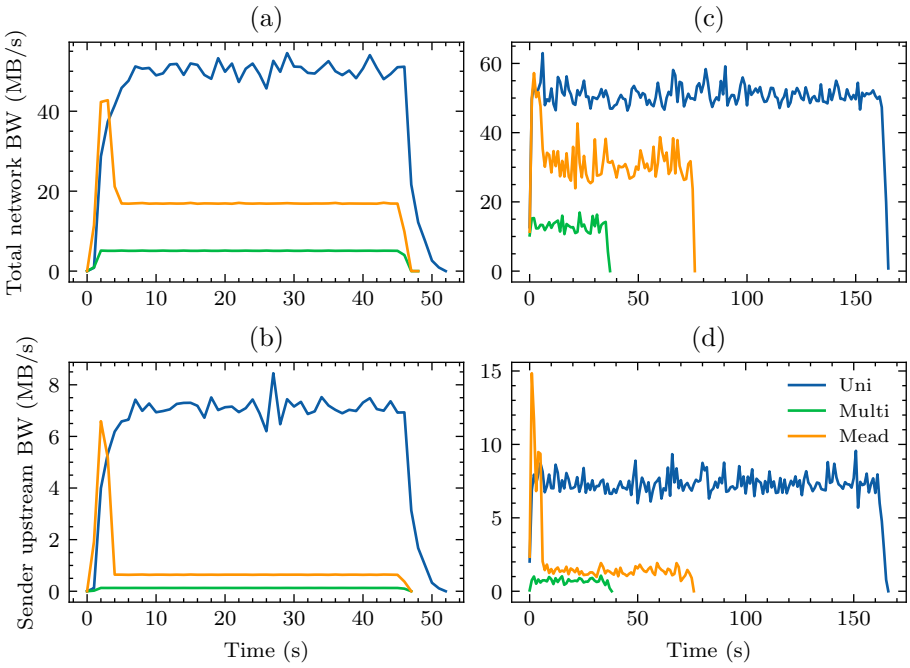

Figure 5.2.: Comparison of network and sender upstream bandwidth utilization. On the left side, **(a)** depicts the total network bandwidth utilization and **(b)** the sender upstream bandwidth utilization during the execution of UC1: Live Stream (1 MB/s for 45s) with 56 clients (5 per router), a maximum of 20 addresses per packet, and a discovery interval of 5 seconds. On the right side, **(c)** represents the total network bandwidth utilization and **(d)** the sender upstream bandwidth utilization during UC2: File Transfer (25 MB) with 45 clients (5 per router), a maximum of 24 addresses per packet and a discovery interval of 5 seconds.

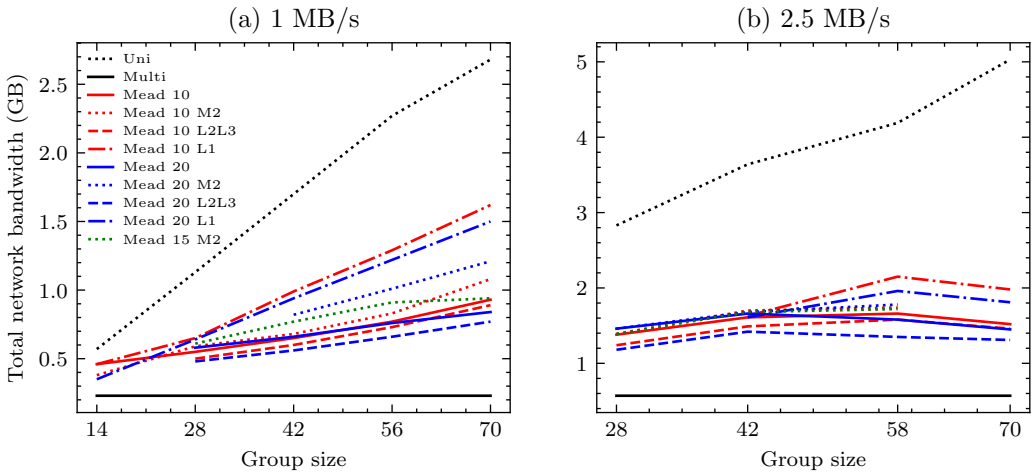

Figure 5.3.: UC1: Live Stream Total network bandwidth utilization. **(1)** Red lines depict measurements with a maximum address list length of 10. **(2)** Blue lines depict measurements with a maximum address list length of 20. **(3)** Line style indicates certain MEADcast configuration, for instance dashed lines represent measurements with network support on layer 2 and 3. **(4)** *Mead 10/20*: Utilizes a maximum address list length of 10 and 20 respectively, with 100% network support. **(5)** *Mead 10/20 M2*: Same as *Mead 10/20* with a merge range of 2. **(6)** *Mead 10/20 L2L3/L1*: Same as *Mead 10/20* with network support limited to layer 2 and 3, and layer 1, respectively.

5.1.2. Grouping and Network support

We conducted tests on MEADcast using various groupings and different levels of MEADcast network support. As previously noted, MEADcast consistently achieved better results in all metrics compared to IP Unicast across all tested groupings and levels of network support. However, we observed variations in our findings depending on the MEADcast configuration.

Bandwidth Utilization

Interestingly, the most significant reduction in total network bandwidth utilization was not achieved with 100% network support, but rather with MEADcast support only on the Distribution (L2) and Access Layer (L3)[1], as shown in Figure 5.3. This was closely followed by measurements with a merge range of 2[2]. In contrast, experiments with MEADcast support only on the Core Layer (L1) achieved the least improvement compared to IP Unicast. Each of these experiments was conducted with a maximum address list length of 10 and 20. Notably, in all tested experiments in UC1, a maximum address list of 20 consistently achieved lower bandwidth utilizations than its counterpart with a limit of 10 addresses. The drop in total network bandwidth utilization in UC1 (2.5 MB/s) is caused by an increased packet loss rate due to the resource limitations of the testbed.

[1]Section 2.2 introduces the terms Core (L1), Distribution (L2), and Access Layer (L3). Subsection 4.4.1 provides information about the corresponding layers in our topology.
[2]Subsection 4.3.2 provides information about the merge range parameter.

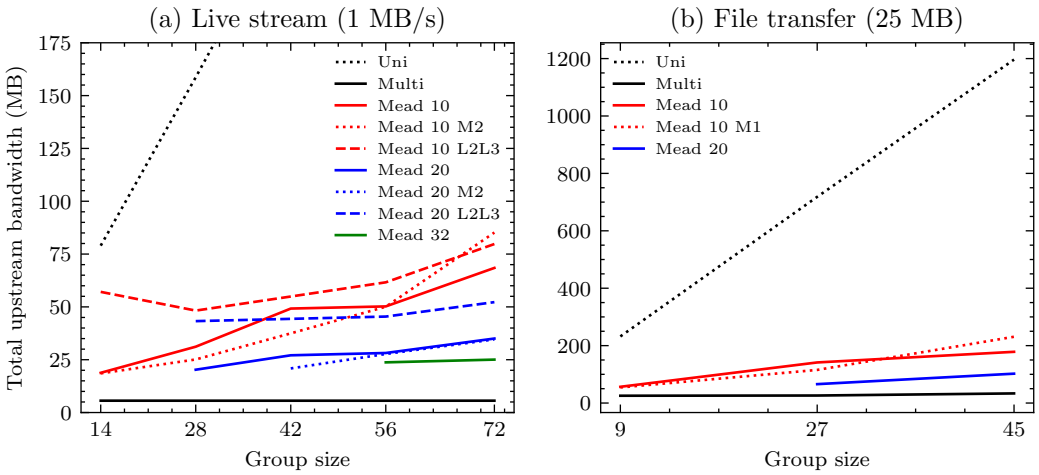

Figure 5.4.: Total sender upstream bandwidth. **(1)** Red lines depict measurements with a maximum address list length of 10. **(2)** Blue lines depict measurements with a maximum address list length of 20. **(3)** Line style indicates certain MEADcast configuration, for instance dashed lines represent measurements with network support on layer 2 and 3. **(4)** *Mead 10/20*: Utilizes a maximum address list length of 10 and 20 respectively, with 100% network support. **(5)** *Mead 10/20 M2*: Same as *Mead 10/20* with a merge range of 2. **(6)** *Mead 10/20 L2L3/L1*: Same as *Mead 10/20* with network support only on layer 2 and 3, and layer 1, respectively.

Figure 5.4 illustrates, that all tested groupings and levels of network support exhibit significantly lower sender upstream bandwidth requirements compared to IP Unicast. Furthermore, as the number of receivers increases, the upstream bandwidth demands of MEADcast grow at a much slower rate than IP Unicast. Additionally, the higher the address list limit, the fewer packets with identical payloads are required, resulting in a greater reduction in sender upstream bandwidth utilization. For instance, configurations with an address limit of 20 consistently achieved lower upstream bandwidth consumption than all configurations with a limit of 10. In contrast to the reductions in total network bandwidth, experiments with MEADcast support exclusively on L2 and L3 achieved the least improvement. Experiments with 100% MEADcast support and a merging range between 1 and 2 yielded comparable results.

Figure 5.5 depicts a comparison of bandwidth usage per link between measurements with 100% MEADcast support and network support limited to L2 and L3. Consistent with the observations regarding total network bandwidth utilization, MEADcast support exclusively on L2 and L3 leads to lower bandwidth consumption per link compared to 100% network support. In both UC1 and UC2, the lower level of MEADcast support notably decreases bandwidth usage on certain links by over 50%.

Furthermore, we executed UC2 with 27 receivers (three per router), a maximum of 30 addresses per packet, and MEADcast support limited to a single router, R_3. This experiment resulted in a reduction in transfer time by 42.98%, network bandwidth utilization by 53.14%, and sender upstream bandwidth utilization by 94.55%.

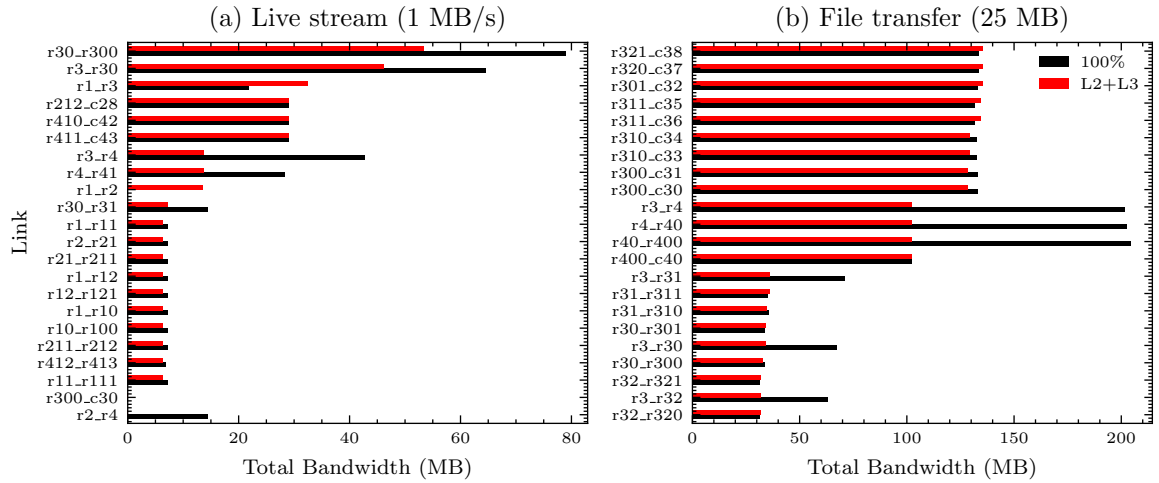

Figure 5.5.: UC2: File Transfer Comparison of total link bandwidth. **(1)** Black bars denote results from measurements with 100% MEADcast support. **(2)** Red bars denote results from measurements with MEADcast support on the Distribution (L2) and Access Layer (L3). **(a)** depicts the results from the execution of UC1: Live Stream (1 MB/s for 45 seconds), with 70 receivers (five per router), and a maximum of 20 addresses per packet. **(b)** depicts the results from the execution of UC2: File Transfer (25 MB), with 45 receivers (five per router), and a maximum of 24 addresses per packet.

Latency

Both the grouping configuration and the level of MEADcast network support have implications for latency. For small group sizes, MEADcast leads to a slight increase in average latency compared to IP Unicast (see Figure 5.6 (a)). However, as the number of receivers grows to 42 (three clients per router), the latency of IP Unicast significantly exceeds that of MEADcast. MEADcast support limited to L2 and L3 exhibits the lowest latency among all tested MEADcast configurations, followed by 100%. Similar to IP Unicast, as the group size increases, MEADcast with periodic discovery phase exceeds the average latency of MEADcast without discovery. However, it still remains lower than IP Unicast. The maximal latency yields similar results, with the latency of MEADcast with discovery phase being similar to or slightly higher than that of IP Unicast (see Figure 5.6 (c)).

The minimal latency of MEADcast is higher than that of IP Unicast (see Figure 5.6 (b)). MEADcast achieves results more akin to IP Multicast. All tested MEADcast configurations with an address list limit of 20 possess lower minimal latency than any configuration with a limit of 10. The increase in latency, as the number of receivers grows, is the highest for the maximal latency and the lowest for the minimal latency. Throughout our experiments, we also measured average, minimal, and maximal Jitter. However, the results indicate no measurable impact from MEADcast on Jitter.

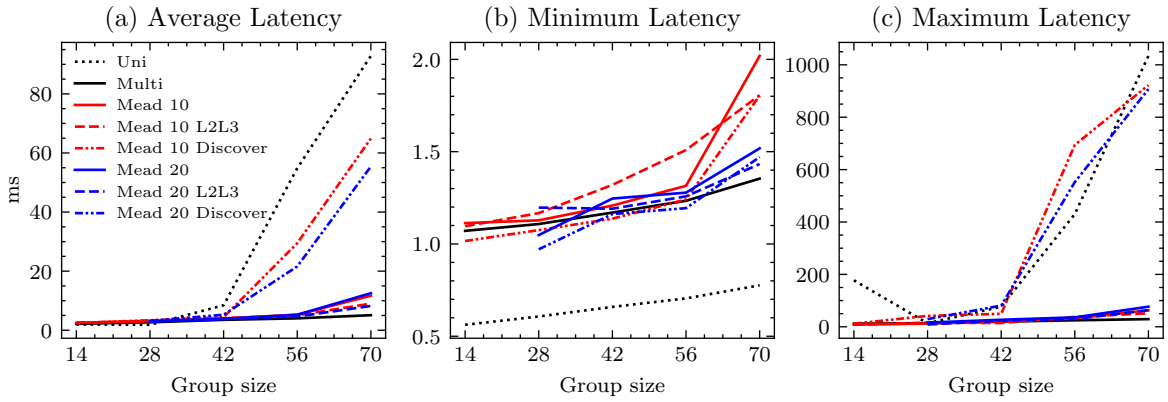

Figure 5.6.: UC1: Live Stream Latency comparison. **(1)** Red lines depict measurements with a maximum address list length of 10. **(2)** Blue lines depict measurements with a maximum address list length of 20. **(3)** Line style indicates certain MEADcast configuration, for instance dashed lines represent measurements with network support on layer 2 and 3. **(4)** *Mead 10/20*: Utilizes a maximum address list length of 10 and 20 respectively, with 100% network support. **(6)** *Mead 10/20 L2L3*: Same as *Mead 10/20* with network support limited to layer 2 and 3.

5.1.3. Effects of the discovery phase

As previously demonstrated, when dealing with a large number of receivers, MEADcast with periodic discovery phase tends to exhibit higher latency compared to MEADcast without discovery phase.

Figure 5.7, illustrates spikes in maximum latency for both 28 and 70 receivers during discovery phases. Additionally, there is a noticeable increase in average latency for 70 clients during these phases, although the increase is marginal for 28 clients. Especially the initial discovery phase results in a significant spike in total network bandwidth utilization, sender upstream bandwidth utilization, and latency (see Figure 5.2 for total network and sender upstream bandwidth utilization, and Figure 5.7 for latency). Moreover, if the network is already operating at its capacity limit, discovery phases, accompanied by an increased traffic volume, can result in elevated packet loss, as shown in Figure 5.8. Notably, the initial discovery phase, accompanied by IP Unicast data transmission, has also the most substantial effect on packet loss.

Consequently, Table 5.4 demonstrates the latency increase caused by the discovery phase, particularly focusing on the initial discovery. In experiments with discovery phase, the average latency increases by 39.57%. Excluding the initial discovery phase and the associated IP Unicast transmission, the increase in latency is reduced by more than half, to 15.69%. Furthermore, the results emphasize a distinct increase in latency as the number of clients rises. For three or fewer clients per router, the latency increase is 11.1% with initial discovery and 3.9% without. However, for more than three clients per router, the latency increase grows to 75.16% with initial discovery and to 30.83% without.

Table 5.5 illustrates the impact of the discovery phase on network bandwidth utilization and sender upstream bandwidth utilization. The discovery phase increases the network bandwidth utilization by an average of 4.69%, while the sender upstream bandwidth uti-

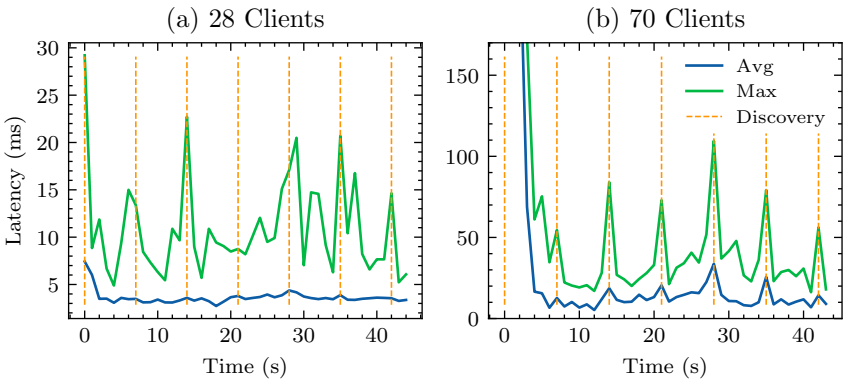

Figure 5.7.: UC1: Live Stream Effect of the discovery phase on latency. The vertical dashed line represents the periodic discovery phase. The green line depicts the maximal latency and the blue line the average latency.

lization increases by 18.36% (compared to the corresponding experiment without discovery phase). Excluding the initial discovery phase and the associated IP Unicast transmission, the increase in network bandwidth utilization reduces to 0.62%, and for sender upstream utilization, it decreases to 0.42%. The theoretically calculated increase in sender upstream bandwidth is merely 0.04%. The average increase in total transfer time is 0.88% for UC2 (25 MB).

Furthermore, we observed that in the current sender implementation, the order in which it receives discovery responses also affects the grouping of the clients. However, we did not observe any performance implications of the change in receiver grouping.

5.1.4. MEADcast in dynamic network environments

In UC3: P2P Video Conference we evaluated MEADcast's resilience and recovery mechanisms required for dynamic network environments.

We deliberately introduced a firewall to drop MEADcast packets during transmission. Figure 5.9 (a) and (b) illustrate, that upon enabling the firewall, receiver bandwidth drops to zero, accompanied by a decrease in network bandwidth utilization. Following the subsequent discovery phase, the sender attempts to assign clients to a router located in front of the firewall. If this is not feasible, the sender falls back to IP Unicast transmission. Consequently, the packets traverse the firewall and the client receives traffic again. Fallback to IP Unicast increases total network bandwidth utilization from 15 MB/s to 50 MB/s, whereas falling back to another MEADcast router raises bandwidth only up to 28 MB/s. Upon firewall deactivation, initially, no change is observed. However, after the subsequent discovery phase, the sender reverts to the initial MEADcast grouping. This results in the network bandwidth utilization returning to its pre-firewall activation level of 15 MB/s.

Furthermore, we assessed MEADcast's resilience to route changes. Despite altering routes during MEADcast transmission, receiver bandwidth remained consistent (see Figure 5.9 (d)). The only observed effect was a peak in latency. After the route change, packet delivery remained undisrupted as we redirected traffic from R_1 to R_3 (*r1_r3*) to traverse via R_4 (*r1_r4*), ensuring the packet keeps traversing all MEADcast routers from its address list (see

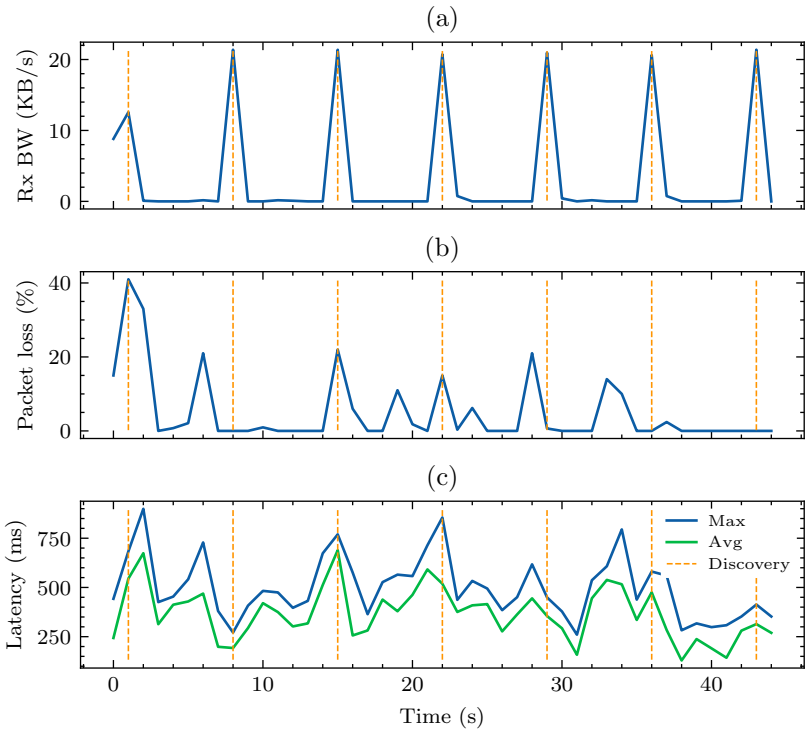

Figure 5.8.: Effect of the discovery phase. **(a)** depicts the received bandwidth on the sender. **(b)** depicts the packet loss on a receiver. **(c)** depicts the maximal latency on a receiver.

		Average Latency *(ms)*				**Rel. increase** *(%)*	
Clients *(per Router)*	**Address limit**	No Dcvr	Incl. init. Dcvr	Excl. init. Dcvr	Diff *(%)*	Incl. init. Dcvr	Excl. init. Dcvr
1	10	2.77	2.72	2.69	0.86%	-2.17%	-3.05%
2	10	3.14	3.81	3.54	6.93%	17.51%	11.37%
3	10	4.31	4.51	4.20	7.01%	4.40%	-2.81%
4	10	5.15	24.16	6.42	73.42%	78.70%	19.86%
5	10	12.42	59.28	33.98	42.67%	79.05%	63.46%
2	20	3.33	3.64	3.50	3.84%	8.46%	4.80%
3	20	4.20	5.77	4.53	21.53%	27.32%	7.39%
4	20	6.10	19.92	6.08	69.46%	69.38%	-0.27%
5	20	9.94	37.49	16.69	55.47%	73.49%	40.47%
				Average	31.24%	39.57%	15.69%

Table 5.4.: Latency increase caused by the discovery phase. We gathered the results by executing the measurements once with and once without periodic discovery phase. The relative difference describes the variation between the measurements conducted with and without the initial discovery phase. The relative increase refers to the corresponding experiment without discovery phase.

		Bandwidth				
		Inclusive initial discovery phase		Exclusive initial discovery phase		
Experiment	**Time**	Network	Sender	Network	Sender	Sender *(Calc.)*
UC1 *(1 MB/s)*	0.00%	6.17%	17.37%	0.54%	-2.24%	0.06%
UC1 *(2.5 MB/s)*	-0.03%	2.79%	16.35%	0.78%	2.48%	0.03%
UC2 *(25 MB)*	0.88%	5.11%	21.37%	0.55%	1.01%	0.02%
Average	0.28%	4.69%	18.36%	0.62%	0.42%	0.04%

Table 5.5.: Bandwidth utilization increase caused by the discovery phase. We gathered the results by executing the measurements once with and once without periodic discovery phase. 100% refers to the corresponding experiment without discovery phase.

Figure 5.9 (c)). It is crucial to note that if a MEADcast packet does not pass through a router listed in its header, there is a risk of incorrect processing, potentially resulting in the packet not being delivered to its intended receivers.

Lastly, we evaluated MEADcast's recovery from a link outages. Immediately after the link failure between R_1 and R_3, receiver bandwidth dropped to zero (see Figure 5.9 (f)). Following the subsequent discovery phase, R_1's output bandwidth (*r1_r3*) increased from 1.5 MB/s to 8 MB/s (see Figure 5.9 (e)). This increase occurred due to a new grouping with an earlier M2U transformation. Once the underlying routing protocol detected the link failure and adjusted the route accordingly, clients resumed receiving traffic. Similar to the route change experiment, the routing protocol redirected traffic from R_1 to R_3 (*r1_r3*) to traverse via R_4 (*r1_r4*), ensuring packets resumed traversing all MEADcast routers from their address list, resulting in correct packet processing and delivery. Following the subsequent discovery phase, the sender reverted to the initial grouping, restoring link bandwidth utilization to its pre-outage level of 1.5 MB/s.

Additionally, we observed instances where a router failed to perform M2U transformation – due to reasons such as an error, MEADcast being disabled, or a routing change – MEADcast packets are forwarded to the client whose address is specified in the IP destination field. This scenario potentially results in a leakage of the addresses of other group members. However, in our experiments, we found that the client software (specifically *"Iperf"* and *"Netcat"*) disregarded the MEADcast header and processed the packet correctly. Furthermore, by injecting discovery responses, we were able to impede MEADcast delivery to multiple receivers and intercept MEADcast packets.

5.2. Discussion

This section evaluates the results from our experiments presented in Section 5.1 and aims to address the research questions formulated in Section 3.3. As outlined in Section 1.3, the primary goal of this thesis is to conduct an evaluation of MEADcast, focusing on its *feasibility*, *performance*, and *potential application domains*. Subsection 5.2.1 discusses the results regarding feasibility, targeting RQ1. Subsection 5.2.2 elaborates on performance

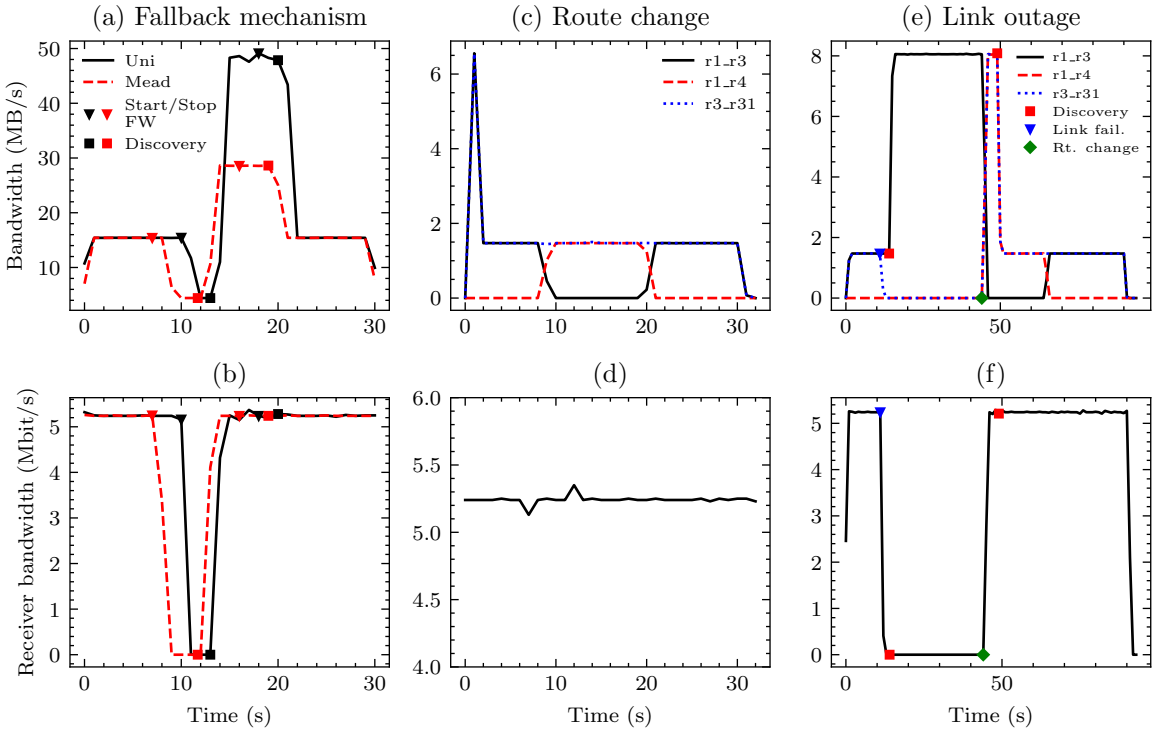

Figure 5.9.: UC3: P2P Video Conference MEADcast fallback and recovery mechanism: **(1)** The left column illustrates the total network bandwidth **(a)** and receiver bandwidth **(b)** in a scenario where MEADcast packets are dropped during transmission, for instance, due to firewall intervention. The black line represents the results when falling back to IP Unicast transmission. The red line illustrates the results from scenarios where the clients could be assigned to a MEADcast router located in front of the firewall. The triangular marker indicate the period during which MEADcast packets are dropped and the square marker represent the discovery phase. **(2)** The middle column illustrates the bandwidth per link **(c)** and receiver bandwidth **(d)** in a scenario where a route change occurred during transmission. **(3)** The right column shows the bandwidth per link **(e)** and receiver bandwidth **(f)** in a scenario where a link outage occurred during transmission. The square marker indicate the discovery phase, the triangular marker represent the time of link outage, and the diamond marker the time of route change.

metrics in comparison to IP Unicast and IP Multicast, answering RQ2. Subsection 5.2.3 deduces use cases for which MEADcast is well-suited, addressing RQ3 and RQ4. Finally, Subsection 5.2.4 proposes several refinements to the MEADcast specification.

5.2.1. Feasibility

RQ1 How robust is the current MEADcast specification? (deployment limitations & structural issues of the protocol specification)

Deployment The conducted series of use cases, encompassing deployment and evaluation, effectively demonstrates the feasibility of employing MEADcast within a medium-sized network. MEADcast deployment requires only the installation of sender and router software. Additionally, the fallback mechanism enables MEADcast to operate even without dedicated router support, presenting a distinct advantage over IP Multicast. In contrast to IP Multicast, which entails increased technical complexity and a compound routing procedure, necessitating all routers to support the protocol (see Subsection 2.1.1), MEADcast imposes no additional requirements beyond the sender and router software. This characteristic facilitates a partial deployment of MEADcast, underscoring its superior feasibility compared to IP Multicast. As we discussed below, this characteristic makes MEADcast particularly suitable in conditions of limited network control.

Robustness MEADcast has demonstrated resilience and recovery from deliberate routing changes, network disruptions such as link and router outages, and packet discarding by intermediate nodes. As MEADcast operates based on IP Unicast routes, its adaptability to evolving network topologies primarily relies on the underlying routing protocol. However, if a packet's route is altered and it does not traverse the routers listed in its header, delivery disruption persists until the next discovery phase. In the event of a firewall dropping packets during MEADcast transmission, disruptions endure until the sender detects the modified topology tree in the subsequent discovery phase. Both MEADcast and IP Unicast fallback effectively address the presence of a firewall. However, reverting to a router positioned in front of the firewall results in 50% less network bandwidth utilization compared to IP Unicast.

Anomaly Handling In cases where routers fail to perform M2U transformation, packets are forwarded to the client specified in the IP destination field, potentially exposing group member IP addresses. Subsection 5.2.4 discusses a potential mitigation preventing the exposure of group information. It is important to note that the handling of anomalous discovery responses is implementation-specific. However, since no authentication mechanism is specified for MEADcast, malicious discovery responses can be injected, potentially hindering transmission to multiple clients and intercepting MEADcast packets. This represents a security threat, as one of MEADcast's objectives is the preservation of endpoint privacy.

Packet Replication The experiments have revealed an inefficiency within the MEADcast routing process. Specially, when multiple router addresses are included within a single packet, sharing a common path of MEADcast routers, the first MEADcast hop generates a replica for each router in the address list. This behavior, while technically correct, arises from the stateless nature of routers, which lack information regarding whether routers from the address list share another intermediate router that could instead perform the replica generation. Although the sender possesses knowledge of how long routers within a packet share the same path, the current MEADcast specification lacks a feature to determine the point of packet replication, thus preventing previous MEADcast routers from doing so. As depicted in Figure 5.5 this inefficiency leads to a significant increase in bandwidth utilization. In Subsection 5.2.4 we propose a feature, which enables the sender to control the point of packet replication.

These results emphasize the feasibility of employing MEADcast in medium-sized networks. Moreover, the experiments illustrate MEADcast's resilience to network disruptions and recovery capabilities. This highlights the protocol's adaptivity and suitability for dynamic network environments, offering promising initial insights into the real-world applicability and resilience of MEADcast. However, anomaly handling is highly implementation-specific. Furthermore, in Subsection 5.2.4 we propose several refinements to the MEADcast specification and further investigation of their implications.

5.2.2. Performance

RQ2 How does MEADcast perform compared to IP Unicast and IP Multicast?

The experiments emphasize that MEADcast significantly enhances performance metrics such as total network bandwidth utilization, sender upstream bandwidth utilization, and total transfer time compared to IP Unicast (see Subsection 5.1.1). On average, MEADcast reduced network bandwidth utilization by more than half (56%), sender upstream bandwidth utilization by 81.23%, and total transfer time by half (UC2: 49.18%). In contrast, compared to IP Multicast, MEADcast increased network bandwidth utilization by 32.42%, sender upstream bandwidth by 15.51%, and total transfer time by 26.97% (UC2).

Subsection 5.1.3 illustrates that the discovery phase produces an average overhead of 4.69% in total network bandwidth utilization, 18.36% in sender upstream bandwidth utilization, and 0.88% (UC2) in total transfer time. However, excluding the initial discovery phase significantly reduces the average overhead to an increase of 0.62% in total network bandwidth utilization, and 0.42% in sender upstream bandwidth utilization.

During most of our measurements, resource utilization peaked, rendering meaningful results for sender and router resource utilization unattainable. Nevertheless, we contend that MEADcast results in reduced resource utilization compared to IP Unicast as the number of receivers increases. Figure 5.6 illustrates a distinct increase in average latency for IP Unicast transmissions with more than 42 group members (three clients per router), coinciding with the onset of packet loss. This suggests, that IP Unicast transmission exhausts the testbed's resources at this point. In contrast, MEADcast maintains constant latency levels with little to no packet loss, attributed to its lower resource utilization. Moreover, the observed increase in average latency for

MEADcast with discovery phase supports our hypothesis. As indicated in Table 5.4, recurring discovery phases entail a minor impact on average latency. In contrast, the initial discovery phase, accompanied by IP Unicast transmission significantly increases latency, network bandwidth utilization, sender upstream bandwidth consumption, and packet loss (see Subsection 5.1.3). This suggests that IP Unicast transmission during the initial discovery phase is the primary cause of resource strain supporting our hypothesis.

The results emphasize the feasibility of a graduate shift from extensive IP Unicast to MEADcast delivery, highlighting significant performance improvements, particularly in terms of reduced total network and sender upstream bandwidth utilization, as well as latency. Furthermore, the overhead generated by the discovery phase primarily originates from the initial discovery, whereas the overhead of subsequent recurring discovery phases is negligible.

5.2.3. Scenario identification

RQ3 Which applications and characteristics are well served by MEADcast?

Across all use cases, MEADcast has consistently demonstrated its benefits.

From small group sizes (< 10) to groups with ≤ 70 receivers, MEADcast has shown improvements in all metrics. As the total number of group members and the number of receivers per router increased, these enhancements became more pronounced. Additionally, MEADcast exhibits performance improvements across all tested session durations. However, due to the overhead associated with the initial discovery phase, we recommend session durations of at least 5 seconds. Particularly with longer session durations, the performance improvements become more significant, as subsequent discovery phases represent a negligible overhead.

MEADcast is well-suited for communication patterns characterized by steady flows (e.g. video/audio stream) and recurring bursts (e.g. file transfer). Although experiments with asymmetric communication patterns were not feasible due to resource limitations, we anticipate that the results from symmetric communication patterns are applicable, except for the enlarged traffic volume inherent in P2P communication.

MEADcast proves beneficial across all group sizes, receiver distributions, session durations, and communication patterns. However, as group sizes, receiver clustering, and session duration increase, the performance improvements compared to IP Unicast become more pronounced. MEADcast excels in throughput-intense scenarios, with no discernible impact on jitter, making it suitable for applications sensitive to jitter fluctuations. However, MEADcast may not be suitable for applications sensitive to initial startup latency.

RQ4 In which conditions is the usage of MEADcast sensible?

The employment of MEADcast proves particularly beneficial when the reduction of the total network bandwidth or sender upstream utilization is a major concern, especially in cases where IP Multicast support cannot be guaranteed. Deploying a single MEADcast router at a strategic location can significantly reduce bandwidth utilization between the sender and router. However, it is important to note that the presence of

a higher number of MEADcast routers may not necessarily translate to performance enhancements, as indicated in Subsection 5.2.1 highlighting packet replication inefficiencies.

MEADcast excels in scenarios characterized by resource constraints and limited network control. Moreover, MEADcast empowers the sender to granularly tailor the traffic pattern according to specific requirements and prevailing network conditions. Given the protocol's support for partial deployment and resilient fallback mechanism, coupled with the absence of any major performance disadvantages, we advocate for the adoption of MEADcast whenever multicast communication is applicable. However, in scenarios where IP Multicast usage can be assured, it remains the preferred choice.

5.2.4. MEADcast Revision Proposal

Drawing from the experience gathered during the implementation and deployment of MEADcast, coupled with the findings of our experiment, we propose several refinements for the next MEADcast revision.

Protocol specification As discussed in Section 4.1, we propose the omission of the Hop-by-Hop IPv6 extension header, which experiences an increased drop rate [Gon+16]. This adjustment aims to reduce the protocol's overhead by eliminating a header that serves no purpose, decrease the likelihood of slow path processing, and increase the probability of being forwarded by non-MEADcast routers. Furthermore, to align with RFC 8200 [DH17] and to mitigate the risk of intermediate nodes discarding MEADcast packets due to a malformed IPv6 routing extension, we advocate for the inclusion of the "Segments Left" field from the static IPv6 routing extension header, with a fixed value of zero, which prevents intermediate nodes that do not recognize the MEADcast header from discarding the packet [DH17]. A detailed exposition of the proposed header specification is provided in Section 4.1.

Anomaly Handling We recommend investigating whether targeting packets to the next MEADcast hop, rather than the first client from the address list, could prevent the exposure of group member information in cases of failure. This investigation should also consider the implications in scenarios of route changes and network disruptions. Since we were able to intercept traffic and impede data transmission to multiple clients, by injecting malicious discovery responses, we suggest the incorporation of an authentication mechanism for discovery responses similar to established routing protocols like OSPF [Moy98].

Packet Replication As discussed in Paragraph 5.2.1, our findings have revealed an inefficiency within the MEADcast routing process, if multiple router addresses within a single packet traverse a shared path of MEADcast routers. To facilitate the sender to prevent the occurrence of premature packet replication, we propose the introduction of a "Don't Replicate" field in the MEADcast header. Drawing inspiration from the IPv6 Hop Limit, this field contains a counter that decrements with each forwarding MEADcast router. As long as the field retains value greater than zero, the packet should not be replicated, facilitating the sender to mitigate this inefficiency. This feature can lead to significant bandwidth savings by determining the point of packet replication.

6. Summary

This chapter concludes this thesis. Section 6.1 provides an summary about this study, while Section 6.2 demonstrate potential avenues for future work.

6.1. Conclusion

In the age of ever-increasing bandwidth consumption, largely driven by multimedia content, network operators and service providers face the challenge of meeting the continuously rising demands. One potential solution is the adoption of multicast delivery. However, the deployment of IP Multicast has fallen short of expectations. Consequently, this research aimed to assess the efficacy of a novel multicast protocol known as MEADcast, focusing on its feasibility, performance, and potential application domains.

A comprehensive evaluation of MEADcast was conducted by implementing its router functionality within the Linux Kernel alongside a standalone implementation of the sender. This evaluation encompassed four distinct use cases, each representing real-world scenarios that potentially benefit from multicast communication. The experimentation unfolded within a medium-sized network comprising 205 nodes. Furthermore, MEADcast was subject to comparison with established alternatives, notably IP Unicast and IP Multicast.

The results highlight the feasibility of deploying MEADcast in medium-sized networks. MEADcast has proven its adaptivity and suitability for dynamic network environments, showcasing resilience and recovery capabilities in response to network disruptions, particularly attributable to its well-functioning fallback mechanism.

Furthermore, the performance comparison emphasized significant performance improvements in comparison to IP Unicast. MEADcast deployment yielded a 56% reduction in network bandwidth utilization, an 81.23% decrease in sender upstream bandwidth utilization, and a 49.18% reduction in total transfer time (UC2). Additionally, overhead generated by the discovery phase primarily originates from its initial discovery, with subsequent recurring discovery phases incurring negligible overhead.

MEADcast exhibits significant improvements across all metrics for applications characterized by communication patterns of steady flows (e.g. video/audio stream) and recurring bursts (e.g. file transfer). Furthermore, MEADcast has proven its efficacy across all group sizes, receiver distributions, and session durations. Notably, as group sizes, receiver clustering, and session duration increase, the performance improvements compared to IP Unicast become more pronounced. MEADcast particularly excels in scenarios demanding high throughput. However, MEADcast may not be suitable for applications with very short session durations or those highly sensitive to initial startup latency.

MEADcast excels in scenarios characterized by resource constraints and limited network control. Deploying a single MEADcast router at a strategic location can significantly reduce bandwidth utilization between the sender and router. However, increasing the number of MEADcast routers may not necessarily enhance performance due to packet replication

inefficiencies. MEADcast empowers the sender to granularly tailor the traffic pattern according to specific requirements and prevailing network conditions. With its resilient fallback mechanism and support for partial deployment, MEADcast adoption is advocated whenever multicast communication is applicable given the absence of major drawbacks. Nonetheless, where IP Multicast usage is assured, it remains the preferred choice.

This thesis stands as a pioneering endeavor, providing a comprehensive evaluation of MEADcast grounded in real-world use cases, diverging from preceding studies exclusively confined to network simulations and SDNs. Furthermore, our study encompasses the inaugural evaluation of MEADcast in its entirety, covering the discovery phase, data delivery phase, transition from IP Unicast to MEADcast delivery, fallback mechanism, receiver grouping, and anomaly handling. Additionally, this study marks the first comparison among IP Unicast, IP Multicast, and MEADcast. Thereby, this study has shown the feasibility of deploying MEADcast in medium-sized networks, offering promising initial insights into the real-world applicability of MEADcast. MEADcast exhibits significant improvements in all metrics across the tested use cases, under various network conditions. Given its resilient fallback mechanism and support for partial deployment, MEADcast adoption is advocated whenever multicast communication is applicable but IP Multicast is unavailable. However, several refinements are proposed for the upcoming MEADcast revision, aiming to address existing inefficiencies and mitigate malicious interference. These refinements include the omission of the Hop-by-Hop IPv6 extension header, integration of the "Segments Left" field from the static IPv6 routing extension header, introduction of a new "Don't Replicate" field, addressing packets to the subsequent MEADcast hop, and implementation of an authentication mechanism for discovery responses.

6.2. Future Work

This thesis illuminated several avenues for future research and development. A natural progression from our current work involves conducting real-world evaluations of MEADcast in moderately controlled environments, such as academic networks. This is particularly relevant given that our experiments were conducted in a virtual environment with para-virtualized network interface. Addressing concerns regarding IPv6 extension header processing [Gon+16], further investigation could explore MEADcast transmission over the internet.

Moreover, as our study did not yield reliable results for P2P communication and group sizes exceeding 100 nodes, future research endeavors should concentrate on addressing these aspects. Optimizing the grouping of receivers into MEADcast packets has emerged as a complex task, warranting exploration into suitable grouping algorithms. Additionally, since MEADcast relies on an out-of-band group membership signaling (join & leave), implementing such mechanisms and investigating the implications of highly dynamic groups require further examination.

Furthermore, the implementation and evaluation of the proposed protocol refinements advocated in our study should be pursued. Additionally, investigating mechanisms to reduce the size of the address list is recommended. Potential approaches could draw inspiration from Xcast extensions such as Xcast+ [Shi+01], which only store the addresses of designated routers in the header, resulting in significant bandwidth savings (see Subsection 2.1.3). Xcast+ stores only the addresses of designated routers in the header. However, it is essen-

tial to consider whether these savings justify the introduction of state management tasks on designated routers.

List of Figures

List of Tables

Acronyms

AS Autonomous system. 29

CT Communication Technology. 1

DFS Depth First Search. 36–38

DoS denial-of-service. 17

DR Designated Router. 10

ICMPv6 Internet Control Message Protocol for IPv6. 8

IGMP Internet Group Management Protocol. 8

ISP Internet Service Provider. 2

KVM Kernel-based Virtual Machine. 44

M2U MEADcast to Unicast. 11, 16, 61, 63

M2X Multicast to Xcast. 10

MEADcast Multicast to Explicit Agnostic Destinations. 2–4, 7, 11–16, 19, 21–27, 29–38, 40, 42–49, 51–53, 55–59, 61–68, 89, 92

MLD Multicast Listener Discovery. 8

MTU Maximum transmission unit. 34

NTP Network Time Protocol. 45

OS Operating System. 23, 44

P2P Peer-to-Peer. 21, 23, 24, 42, 43, 47, 65, 68

PTP Precision Time Protocol. 45

QoS Quality of Service. 18

RP Rendezvous Point. 8

RPF Reverse Path Forwarding. 9

SDN Software-defined networking. 3, 68

SDU Service Data Unit. 24

SPT Shortest Path Tree. 8

SSM Source-specific Multicast. 8, 47, 88

TCP Transmission Control Protocol. 46

UDP User Datagram Protocol. 14, 48

VM Virtual Machine. 44, 45

VoIP Voice over IP. 20

X2M Xcast to Multicast. 10

X2U Xcast to Unicast. 9, 10

Xcast Explicit Multicast. 4, 7, 9–11, 68

Xcast+ Explicit Multicast Extension. 4, 7, 10, 68

Bibliography

[88] *Distance Vector Multicast Routing Protocol*. RFC 1075. Nov. 1988. DOI: 10.17487/RFC1075. URL: https://www.rfc-editor.org/info/rfc1075.

[Aca14] Cisco Networking Academy. *Connecting Networks. Companion Guide*. 1st ed. Cisco Press, May 2014. ISBN: 978-1-58713-332-9.

[AE15] Anders SG Andrae and Tomas Edler. "On global electricity usage of communication technology: trends to 2030". In: *Challenges* 6.1 (2015), pp. 117–157.

[Aru19] Aruba. *Aruba 2530 Multicast and Routing Guide for ArubaOS-Switch 16.09*. 1st ed. June 2019. URL: https://www.arubanetworks.com/techdocs/AOS-Switch/16.09/Aruba%202530%20Multicast%20and%20Routing%20Guide%20for%20AOS-S%20Switch%2016.09.pdf (visited on 09/20/2023).

[Aut24] Internet Assigned Numbers Authority. *Protocol Numbers*. Internet Assigned Numbers Authority, Jan. 8, 2024. URL: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml (visited on 02/15/2024).

[Bas04a] Doug Baskins. *judy(3) - Linux man page*. die.net, Oct. 2004. URL: https://linux.die.net/man/3/judy (visited on 02/20/2024).

[Bas04b] Doug Baskins. *What is Judy?* Judy Team, Oct. 2004. URL: https://judy.sourceforge.net/index.html (visited on 02/20/2024).

[BGC04] Ali Boudani, Alexandre Guitton, and Bernard Cousin. "GXcast: Generalized explicit multicast routing protocol". In: *Proceedings. ISCC 2004. Ninth International Symposium on Computers And Communications (IEEE Cat. No. 04TH8769)*. Vol. 2. IEEE. 2004, pp. 1012–1017.

[Boi+07] Rick Boivie et al. *Explicit multicast (Xcast) concepts and options*. Tech. rep. 2007.

[Cai+02] Bradley Cain et al. *Internet Group Management Protocol, Version 3*. RFC 3376. Oct. 2002. DOI: 10.17487/RFC3376. URL: https://www.rfc-editor.org/info/rfc3376.

[Car21] Cartesian. *US-Broadband-Household-Bandwidth-Demand-Study*. July 2021. URL: https://www.cartesian.com/wp-content/uploads/2021/07/Cartesian_NCTA-US-Broadband-Household-Bandwidth-Demand-Study-July-2021.pdf (visited on 09/19/2023).

[Cha+07] Ravi Chandra et al. *Multiprotocol Extensions for BGP-4*. RFC 4760. Jan. 2007. DOI: 10.17487/RFC4760. URL: https://www.rfc-editor.org/info/rfc4760.

[Cis] Cisco. "Configuring MLD Snooping for IPv6 Multicast Traffic". In: *Cisco IOS Software Configuration Guide*. Vol. 12.2. Cisco. Chap. 30, 301–3016. URL: https://www.cisco.com/en/US/docs/general/Test/dwerblo/broken_guide/snoopmld.html#wp1076709 (visited on 04/11/2024).

[Cis01] Cisco. *IP Multicast Technology Overview*. Cisco, Oct. 2001. URL: https://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/ip_multicast/White_papers/mcst_ovr.html (visited on 10/24/2023).

[Cis08] Inc. Cisco Systems. *Enterprise Campus 3.0 Architecture: Overview and Framework*. Cisco Systems, Inc., Apr. 2008. URL: https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Campus/campover.html (visited on 01/10/2024).

[Cis14] Inc. Cisco Systems. *Campus Wired LAN Technology Design Guide*. Cisco Systems, Inc., Aug. 2014. URL: https://www.cisco.com/c/dam/en/us/td/docs/solutions/CVD/Aug2014/CVD-CampusWiredLANDesignGuide-AUG14.pdf (visited on 01/10/2024).

[CJ13] Brian E. Carpenter and Sheng Jiang. *Transmission and Processing of IPv6 Extension Headers*. RFC 7045. Dec. 2013. DOI: 10.17487/RFC7045. URL: https://www.rfc-editor.org/info/rfc7045.

[CK13] Stuart Cheshire and Marc Krochmal. *Multicast DNS*. RFC 6762. Feb. 2013. DOI: 10.17487/RFC6762. URL: https://www.rfc-editor.org/info/rfc6762.

[Clo23] Cloudflare. *Cloudflare Radar Internet traffic trends*. Cloudflare, 2023. URL: https://radar.cloudflare.com/traffic (visited on 09/20/2023).

[CV04] Luis Costa and Rolland Vida. *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*. RFC 3810. June 2004. DOI: 10.17487/RFC3810. URL: https://www.rfc-editor.org/info/rfc3810.

[DC90] Stephen E Deering and David R Cheriton. "Multicast routing in datagram internetworks and extended LANs". In: *ACM Transactions on Computer Systems (TOCS)* 8.2 (1990), pp. 85–110.

[Dee86] Dr. Steve E. Deering. *Host extensions for IP multicasting*. RFC 988. July 1986. DOI: 10.17487/RFC0988. URL: https://www.rfc-editor.org/info/rfc988.

[Dee89] Dr. Steve E. Deering. *Host extensions for IP multicasting*. RFC 1112. Aug. 1989. DOI: 10.17487/RFC1112. URL: https://www.rfc-editor.org/info/rfc1112.

[DH06] Dr. Steve E. Deering and Bob Hinden. *IP Version 6 Addressing Architecture*. RFC 4291. Feb. 2006. DOI: 10.17487/RFC4291. URL: https://www.rfc-editor.org/info/rfc4291.

[DH17] Dr. Steve E. Deering and Bob Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 8200. July 2017. DOI: 10.17487/RFC8200. URL: https://www.rfc-editor.org/info/rfc8200.

[Dio+00] Christophe Diot et al. "Deployment issues for the IP multicast service and architecture". In: *IEEE network* 14.1 (2000), pp. 78–88.

[DT19] Vitalian Danciu and Cuong Ngoc Tran. "MEADcast: Explicit Multicast with Privacy Aspects". In: *International Journal on Advances in Security* 12.1 & 2 (2019), pp. 13–28. ISSN: 1942-2636.

[Fen+16] Bill Fenner et al. *Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)*. RFC 7761. Mar. 2016. DOI: 10.17487/RFC7761. URL: https://www.rfc-editor.org/info/rfc7761.

[GC06] Mukesh Gupta and Alex Conta. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. RFC 4443. Mar. 2006. DOI: 10.17487/RFC4443. URL: https://www.rfc-editor.org/info/rfc4443.

[GL22] Fernando Gont and Will (Shucheng) LIU. *Recommendations on the Filtering of IPv6 Packets Containing IPv6 Extension Headers at Transit Routers*. RFC 9288. Aug. 2022. DOI: 10.17487/RFC9288. URL: https://www.rfc-editor.org/info/rfc9288.

[Gon+16] Fernando Gont et al. *Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World*. RFC 7872. June 2016. DOI: 10.17487/RFC7872. URL: https://www.rfc-editor.org/info/rfc7872.

[Gon+21] Fernando Gont et al. *Operational Implications of IPv6 Packets with Extension Headers*. RFC 9098. Sept. 2021. DOI: 10.17487/RFC9098. URL: https://www.rfc-editor.org/info/rfc9098.

[Har+07] Szabolcs Harcsik et al. "Latency evaluation of networking mechanisms for game traffic". In: *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*. 2007, pp. 129–134.

[Hav98] Geoff Haviland. "Designing High-Performance Campus Intranets with Multilayer Switching". In: *White Paper, Cisco Systems, Inc* (1998). URL: https://web.archive.org/web/20000903190239/http://www.cisco.com/warp/public/cc/so/cuso/epso/entdes/highd_wp.pdf (visited on 01/12/2024).

[Hua23] Ltd. Huawei Technologies Co. "Typical Networking Architectures for Campus Networks and Case Practice". In: *Data Communications and Network Technologies*. Singapore: Springer Nature Singapore, 2023, pp. 471–500. ISBN: 978-981-19-3029-4. DOI: 10.1007/978-981-19-3029-4_15. URL: https://doi.org/10.1007/978-981-19-3029-4_15 (visited on 01/10/2024).

[IET21] IETF. *QUIC in the Internet industry*. IETF, June 3, 2021. URL: https://www.ietf.org/blog/quic-industry/ (visited on 09/23/2023).

[IT21] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021. DOI: 10.17487/RFC9000. URL: https://www.rfc-editor.org/info/rfc9000.

[ITU23] ITU. *Measuring digital development Facts and Figures 2022*. 2023. URL: https://www.itu.int/itu-d/reports/statistics/facts-figures-2022/ (visited on 09/17/2023).

[JC20] Matt Joras and Yang Chi. *How Facebook is bringing QUIC to billions*. Meta, Oct. 21, 2020. URL: https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/ (visited on 09/23/2023).

[Ker23a] Michael Kerrisk. *epoll(7) — Linux manual page*. man7.org, Dec. 2023. URL: https://www.man7.org/linux/man-pages/man7/epoll.7.html (visited on 02/20/2024).

[Ker23b] Michael Kerrisk. *timerfd_create(2) — Linux manual page*. man7.org, Dec. 2023. URL: https://www.man7.org/linux/man-pages/man2/timerfd_create.2.html (visited on 02/20/2024).

[KPP93] Vachaspathi P. Kompella, Joseph C. Pasquale, and George C. Polyzos. "Multicast routing for multimedia communication". In: *IEEE/ACM transactions on Networking* 1.3 (1993), pp. 286–292.

[LN93] Xiaola Lin and Lionel M. Ni. "Multicast communication in multicomputer networks". In: *IEEE transactions on Parallel and Distributed Systems* 4.10 (1993), pp. 1105–1117.

[Mal98] Gary S. Malkin. *RIP Version 2*. RFC 2453. Nov. 1998. DOI: 10.17487/RFC2453. URL: https://www.rfc-editor.org/info/rfc2453.

[MF03] David Meyer and Bill Fenner. *Multicast Source Discovery Protocol (MSDP)*. RFC 3618. Oct. 2003. DOI: 10.17487/RFC3618. URL: https://www.rfc-editor.org/info/rfc3618.

[Mog84] J.C. Mogul. *Broadcasting Internet datagrams in the presence of subnets*. RFC 922. Oct. 1984. DOI: 10.17487/RFC0922. URL: https://www.rfc-editor.org/info/rfc922.

[Moy94] John Moy. *MOSPF: Analysis and Experience*. RFC 1585. Mar. 1994. DOI: 10.17487/RFC1585. URL: https://www.rfc-editor.org/info/rfc1585.

[Moy98] John Moy. *OSPF Version 2*. RFC 2328. Apr. 1998. DOI: 10.17487/RFC2328. URL: https://www.rfc-editor.org/info/rfc2328.

[Nar04] Dr. Thomas Narten. *Assigning Experimental and Testing Numbers Considered Useful*. RFC 3692. Jan. 2004. DOI: 10.17487/RFC3692. URL: https://www.rfc-editor.org/info/rfc3692.

[NAS05] Jonathan Nicholas, Andrew Adams, and William Siadak. *Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)*. RFC 3973. Jan. 2005. DOI: 10.17487/RFC3973. URL: https://www.rfc-editor.org/info/rfc3973.

[Ngu19] Duc Minh Nguyen. "Deployment of MEADcast in Stub Software-Defined Networks". Ludwig-Maximilians-University of Munich, Mar. 31, 2019.

[Pos81] Jon Postel. *Internet Protocol*. RFC 791. Sept. 1981. DOI: 10.17487/RFC0791. URL: https://www.rfc-editor.org/info/rfc791.

[Pro] FRRouting Project. *FRR Documentation*. URL: https://frrouting.org/doc/ (visited on 03/23/2024).

[RES06] Sylvia Ratnasamy, Andrey Ermolinskiy, and Scott Shenker. "Revisiting IP multicast". In: *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. 2006, pp. 15–26.

[RH03] Sandro Rafaeli and David Hutchison. "A survey of key management for secure group communication". In: *ACM Computing Surveys (CSUR)* 35.3 (2003), pp. 309–329.

[Ric15] Mark Richards. *Software architecture patterns*. 2015.

[Rij94] Anil Rijsinghani. *Computation of the Internet Checksum via Incremental Update*. RFC 1624. May 1994. DOI: 10.17487/RFC1624. URL: https://www.rfc-editor.org/info/rfc1624.

[Shi+01] Myung-Ki Shin et al. “Explicit multicast extension (Xcast+) for efficient multicast packet delivery”. In: *ETRI journal* 23.4 (2001), pp. 202–204.

[Sim+07] William A. Simpson et al. *Neighbor Discovery for IP version 6 (IPv6)*. RFC 4861. Sept. 2007. DOI: 10.17487/RFC4861. URL: https://www.rfc-editor.org/info/rfc4861.

[ST02] Sherlia Y Shi and Jonathan S Turner. “Routing in overlay multicast networks”. In: *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 3. IEEE. 2002, pp. 1200–1208.

[TD18] Cuong Ngoc Tran and Vitalian Danciu. “Privacy-Preserving Multicast to Explicit Agnostic Destinations”. In: *Proceedings of the Eighth International Conference on Advanced Communications and Computation (INFOCOMP 2018)*. 2018, pp. 60–65.

[Zha+06] Beichuan Zhang et al. “Universal IP multicast delivery”. In: *Computer Networks* 50.6 (2006), pp. 781–806.

[Zim80] Huber Zimmermann. “OSI reference model-the ISO model of architecture for open systems interconnection”. In: *IEEE Transactions on communications* 28.4 (1980), pp. 425–432.

Appendices

A. Code

```
sudo python3 mk_topo.py \
    -t "template/testbed.json" \
    -o "out/" \
    -n "template/net.xml" \
    -u user \
    --ssh "~/id_rsa_vms.pub" \
    --client-run "template/run.sh template/run_client.sh" \
    --router-run "template/run.sh template/run_router.sh" \
    --backing-file "/var/lib/libvirt/images/templates/debian-12-nocloud-amd64.raw"
    ↪  \
    -c "template/chrony.conf" \
    -f "template/frr.conf" \
    -k kernel=/var/lib/libvirt/images/kernel/vmlinuz-6.5.2-dirty,
    ↪  initrd=/var/lib/libvirt/images/kernel/initrd.img-6.5.2-dirty,
    ↪  kernel_args="root=/dev/vda1 ro console=ttyS0,115200" \
    --os-variant debian10
```

Listing A.1.: Build topology start command

```
{
    "stub_range": "fd14:0::/32",
    "tran_range": "fd14:0::/32",
    "mgm_gw": "10.10.0.1/8",
    "stubs": [
        {
            "id": 10,
            "mask": 112,
            "area": 1,
            "router": 10,
            "clients": 5
        }, {
            "id": 11,
            "mask": 112,
            "area": 1,
            "router": 10,
            "clients": 5
        }, {
            "id": 20,
            "mask": 112,
            "area": 2,
            "router": 20,
            "clients": 5
        }, {
            "id": 21,
            "mask": 112,
            "area": 2,
            "router": 20,
            "clients": 5
        }
    ],
    "trans": [
        [[  1,   2], 0],  /* [[ <ROUTER ID>, <ROUTER ID>], <OSPF AREA>] */
        [[  1,  10], 1],
        [[  2,  20], 2],
    ],
    "area_boundary": [
        /* { <ROUTER ID>, <OSPF AREA>, <PUBLISHED NET MASK> } */
        { "id": 1, "area": 1, "mask": 96 },
        { "id": 2, "area": 2, "mask": 96 },
    ]
}
```

Listing A.2.: Network topology configuration file format. The resulting topology is illustrated in Figure 4.12

```
Usage: mdc_send [OPTIONS...] [PEER ADDRESSES] [PEER PORT]
MEADcast sender

 Network:
  -a, --address=ipv6         Specify source address.
  -i, --interface=ifname     Specify interface to use.
  -p, --port=port            Specify source port.
      --tun-address=ipv6     Specify host route to send traffic to.
  -t, --tun=ifname           Specify name of the created TUN device.

 Discovery:
  -d, --delay=secs           Specify delay until initial discovery phase.
  -I, --interval=secs        Specify discovery interval.
  -T, --timeout=secs         Specify discovery timeout.
  -w, --wait                 If specified, sender waits for traffic before
                             starting discovery phase.

 Grouping:
      --max=max              Specify the maximal number of addresses per
                             MEADcast packet.
      --min-leaves=leaves    Routers with less leaves than `leaves` and less
                             routers than `routers` get removed from tree.
      --min-routers=routers  See `min-leaves`.
  -m, --merge=distance       Specify whether to merge leaves with distinct
                             parent routers under an common ancestor.
                             `distance` determines the range within which leaves
                             will be merged. Assigning `distance` a value of 0
                             disables this feature.
      --ok=ok                Specify whether a packet can be finished
                             prematurely if it contains an equal or greater
                             number of addresses than `ok`. Assigning `ok` a
                             value greater or equal than `max` disables this
                             feature.
  -s, --split                Specify whether leaves of same parent can be split
                             into multiple packets. If a router has more leaves
                             than the maximal number of addresses per packet,
                             they will be split regardless of this flag.

 Verbosity
      --print-groups         Prints grouping to stdout
      --print-tree           Prints topology tree to stdout

  -?, --help                 Give this help list
      --usage                Give a short usage message
  -V, --version              Print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.
```

Listing A.3.: MEADcast sender help prompt

```
sudo ip6tables -A INPUT -i enp2s0 -j DROP
sudo ip6tables -A FORWARD -i enp2s0 -j DROP
```

Listing A.4.: Simulate link failure between R1 and R3 by dropping all packets on R3

```
# enable
sudo ip6tables -A INPUT -i enp2s0 -m ipv6header --header ipv6-route --soft -j DROP
sudo ip6tables -A FORWARD -i enp2s0 -m ipv6header --header ipv6-route --soft -j
↪  DROP

# disable
sudo ip6tables -F
```

Listing A.5.: Enable/Disable firewall on R3

```
HOSTS="/home/rnp/experiments/ex1_stream/util/s1_rx"; \
    ~/rx $HOSTS start && \
    ssh -i ~/id_rsa_vms user@10.10.30.1 'time parallel -v -a s1_tx -j $(cat s1_tx |
    ↪  wc -l) iperf -c {} -u -V -e --no-udp-fin -b 5000k -t 45' && \
    sleep 3 && \
    ~/rx $HOSTS stop ~/experiments/ex1_stream/s1/res/uni/5_45s/  > /dev/null
```

Listing A.6.: Start command IP Unicast measurement. The `"rx start"` command connects to each address in `"$HOSTS"` and starts Iperf in server mode. The sender 10.10.30.1 transmits 5 Mbit/s for 45 seconds to all addresses in `"s1_tx"`. The `"rx stop"` command connects to each receiver, stops Iperf and copies the result to the destination directory.

```
HOSTS="/home/rnp/experiments/ex1_stream/util/s5_rx"; \
    ~/rx $HOSTS start multi fd14::3:30:1  && \
    ssh -i ~/id_rsa_vms user@10.10.30.1 'time iperf -c [ff3e::4321:1234]%enp2s0 -T
    ↪  32 -u -V -e --no-udp-fin -b 2500k -t 45' && \
    sleep 3 && \
    ~/rx $HOSTS stop ~/experiments/ex1_stream/s5/res/multi/3_45s/ > /dev/null
```

Listing A.7.: Start command IP Muticast measurement. The `"rx"` script connects to each address in `"$HOSTS"` and starts Iperf in server mode listening in SSM to `"fd14::3:30:1"`. The sender 10.10.30.1 transmits 2.5 Mbit/s for 45 seconds via IP Multicast. The `"rx stop"` command connects to each receiver, stops Iperf and copies the result to the destination directory.

```
HOSTS="/home/rnp/experiments/ex1_stream/util/s5_rx"; \
    ~/rx $HOSTS start && \
    ssh -i ~/id_rsa_vms user@10.10.30.1 "time iperf -c fd15::1 -u -V -e
    ↪  --no-udp-fin -b 1000k -t 45 -l $(echo '1484 - 20 * 16 - 52' | bc)" && \
    sleep 3 && \
    ~/rx $HOSTS stop ~/experiments/ex1_stream/s5/res/mead/1_45s/merge/2_20/ >
    ↪  /dev/null
```

Listing A.8.: Start command MEADcast measurement. The `"rx"` script connects to each address in `"$HOSTS"` and starts Iperf in server mode. The sender 10.10.30.1 transmits 1 Mbit/s for 45 seconds to `"fd15::1"`, which is usually the address of the TUN interface. The `"rx stop"` command connects to each receiver, stops Iperf and copies the result to the destination directory.

```
# enable / disable MEADcast
echo 1 > /proc/sys/net/ipv6/meadcast/enable

# set minimum destination count for premature M2U
echo 3 > /proc/sys/net/ipv6/meadcast/min_dsts
```

Listing A.9.: MEADcast router configuration

B. Results

Mode	Vol. *(MB)*	Time	Bandwidth	
			Network	Sender
Multi	25	61.54%	64.69%	89.03%
Multi	100	61.66%	64.29%	89.07%
Mead 10	25	22.92%	29.46%	75.62%
Mead 10	100	22.10%	28.42%	75.30%
Mead 10, Merge 1	25	41.19%	46.82%	76.59%
Mead 10, Merge 1	100	40.27%	46.22%	75.82%
Mead 10, Layer 1	25	49.71%	50.05%	87.31%
Mead 10, Layer 1	100	35.26%	49.80%	87.38%

Table B.1.: UC2: File Transfer Relative savings in comparison to unicast. The results highlight, that the relative savings are analog between the 25MB and 100MB experiment. Based on this observation, we continued the experiments only with a data volume of 25MB. Further experiments are required, to investigate whether this conclusion is applicable for other scenarios.

		BW Reduction		**Ranking**	
Clients	Mode	Network	Sender	Network	Sender
1	100%	19.15%	76.19%	3	2
1	Merge	32.52%	**76.54%**	2	1
1	L2L3	19.04%	27.69%	4	4
1	L1	**38.58%**	69.84%	1	3
2	100%	51.29%	80.36%	2	2
2	Merge	48.20%	**84.17%**	3	1
2	L2L3	**55.82%**	69.59%	1	3
2	L1	42.31%	69.65%	4	4
3	100%	62.02%	79.57%	2	2
3	Merge	59.90%	**84.44%**	3	1
3	L2L3	**64.85%**	77.20%	1	3
3	L1	41.96%	70.63%	4	4
4	100%	65.85%	84.09%	2	2
4	Merge	63.52%	**84.14%**	3	1
4	L2L3	**67.97%**	80.48%	1	3
4	L1	43.05%	71.31%	4	4
5	100%	65.44%	**79.40%**	2	1
5	Merge	59.53%	74.36%	3	3
5	L2L3	**66.72%**	76.00%	1	2
5	L1	39.47%	66.27%	4	4

Table B.2.: Performance comparison of MEADcast configurations