IPv6 multicast home proxy

ELIS KULLBERG
and
HANNES JUNNILA

KTH Information and
Communication Technology

Bachelor of Science Thesis
Stockholm, Sweden 2010

TRITA-ICT-EX-2010:106

Kungliga Tekniska Högskolan
Royal Institute of Technology

Version:1.1
Date: 3/6/2010

IPv6 multicast home proxy

Bachelor's thesis

Elis Kullberg and Hannes Junnila
elisk@kth.se & haju@kth.se

Supervisor and examiner: Gerald Q. Maguire Jr.

Abstract

The Internet is becoming increasingly fragmented, leading to a more heterogeneous end-user experience depending on the user's network location (i.e., point of attachment to the network). This is a consequence of several ongoing changes of the Internet. Different regions of the world are in different phases of their rollout of IPv6, making intercommunication increasingly challenging. Copyright legislation has caught up with ICT technology, but differences in licensing agreements may very from nation to nation which often hinders content being accessed beyond borders. Finally, several high-profile government attempts have been made to enforce stringent censorship of data.

Therefore, we believe that a demand exists for simple consumer-oriented technologies for proxying and tunneling data between separate regions of the Internet. Furthermore, we believe that this demand will increase dramatically during the coming years. A key success factor for this next generation of proxies will be the ability to handle multicast IPv6 packets, as these packets represent the most probable distribution method for IPTV in the future.

This thesis examines the challenges presented by IPv6 multicast-routing in the context of constructing a proxy. It also presents a best-practice solution to the problem of designing, implementing, and utilizing such a proxy. The thesis also contains a review of current IPv6 multicast routing technology. Several implementations are benchmarked against each other, with the goal of building a prototype for a consumer-oriented IPv6 multicast proxy. The prototype is presented and was tested. These tests demonstrate the functionality of the prototype proxy and reveal areas where the prototype could be improved. Finally a possible capitalization strategy is suggested.

Sammanfattning

Internet utvecklas mot att bli mer fragmenterat. Detta leder till en heterogen användarupplevelse beroende på uppkopplingspunkt. Utvecklingen är en konsekvens av flera pågående trender. Världens olika regioner ligger i ofas i utbyggnaden av IPv6 vilket medför nya tekniska utmaningar. Samtidigt har upphovsrättslagstiftningen hunnit ikapp teknikutvecklingen, så att länder med olika licensieringsmodeller inte kan dela innehåll. Slutligen försöker flera länder aktivt censurera datatrafik.

Som konsekvens av detta ökar behovet för enkla konsumentorienterade metoder för att knyta ihop olika delar av Internet, så att åtkomst till data garanteras oavsett uppkopplingspunkt. Därmed förutspår vi att efterfrågan på produkter baserade på sofistikerad tunnelteknik kommer öka under de kommande åren.

Denna rapport undersöker de utmaningar IPv6 multicast routing medför i samband med byggandet av en IPv6 multicast proxy. Rapporten presenterar en grundlig teoretisk genomgång av tekniken bakom IPv6 multicast routing. Vidare föreslås ett optimalt tillvägagångssätt för att designa, bygga och använda en sådan proxy. Flera existerande tekniker för multicast forwarding utvärderas och jämförs. Utifrån utvärderingen byggdes tre implementeringar av en IPv6 multicast proxy. Därefter analyseras dessa, tillsammans med förslag för fortsatta studier. Slutligen presenteras en möjlig kapitaliseringsstrategi för tekniken.

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

AICCU	Automatic IPv6 Connectivity Client Utility
ARP	Address resolution protocol
AS	Autonomous System
AYIYA	Anything in anything
CBT	Core based tree
DHCP	Dynamic host configuration protocol
DVMRP	Distance vector mode, reverse path
ECMH	Easy Cast du Multi Hub
GID	Group identifier
GRE	Generic Routing Encapsulation
IANA	International Assigned Numbers Authority
ICMP	Internet Control Message Protocol
ICMPv6	Internet Control Message Protocol, version 6
ID	Identifier
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPTV	Internet Protocol television
IPv4	Internet Protocol, version four
IPv6	Internet Protocol, version six
ISP	Internet Service Provider
KVM	Kernel Virtual Machine
MAC	Media Access Control
MLD	Multicast Listener Discovery
MOSPF	Multiple Open Shortest Path First
MRIB	Multicast Routing Information Base
NAT	Network Address Translation
NDP	Neighbor Discovery Protocol
PIM	Protocol Independent Multicast
PIM-DM	Protocol Independent Multicast - Dense Mode
PIM-SM	Protocol Independent Multicast - Sparse Mode
PIM-SSM	Protocol Independent Multicast - Source Specific Multicast
RFC	Request For Comments
RIID	Rendezvous Point Interface Identifier
RP	Rendezvous Point
RPF	Reverse Path Forwarding
SCTP	Stream Control Transmission Protocol
SPT	Shortest Path Tree
TCP	Transmission Control Protocol
TIC	Tunnel Information and Control protocol
TTL	Time To Live
TV	Television
UDP	User Datagram Protocol

UPnP	Universal Plug and Play
VLC	VideoLAN Client
XORP	eXtensible Open Router Platform

Acknowledgements

We would like to thank our examiner and supervisor Gerald Q. “Chip” Maguie Jr. for his valuable support and feed-back. We would also like to thank Micha Lenk, the maintainer of smcroute, for his technical support.

We would also like to express our gratitude to all our fellow students who have inspired us during the project.

1. Introduction

In recent years there has been much discussion about the shortage of unassigned IPv4 addresses. Newly industrialized countries have become the focal point of this issue because of their growing internet usage and their limited number of assigned addresses [NARTEN]. The problem is even worse within the multicast ranges of the IPv4 address space, since only a fraction is administered by IANA [RFC5771]. For regional service providers, the situation has been alleviated by technologies such source-specific multicast (see Section 3.3.2). However, these fixes limit the usability of multicast in current applications.

The worldwide introduction of IPv6 will help to alleviate these problems by providing a much larger address space for multicast traffic. More IANA-specified permanent addresses will be available. There will also be a clear difference between permanent group identifications and temporary group identifications (see Section 3.3). Due to the more efficient management of multicast, we expect a much more extensive use of multicast as a distribution platform in a wide range of contexts.

Unfortunately, due to the difficulties experienced in IPv6 rollouts, it cannot be expected that all hosts will simultaneously migrate to IPv6 [DURAND]. Asia has taken the lead in terms of the number of IPv6 roll-outs. As a sign of how important the transition to IPv6 is, the rollout of IPv6 is included in the latest 5-year plan for China [CNGI]. The European Union and the United States have little economic incentive to migrate at the moment, implying that there is a risk for a more fragmented Internet in the future. Tunneling IPv6-traffic through the legacy infrastructure will therefore be crucial for the survival of a coherent end-user web (and other service) experience [DURAND].

We have found two main issues that need to be addressed in order to provide a feasible migration strategy to a global IPv6-infrastructure. Firstly tunneling will play a crucial role as different parts of the world migrate to IPv6 at their own pace. In Section 3.2 we will review current and potential tunneling protocols in order to assess their potential to interconnect a patchwork of isolated IPv6 sites. Secondly, transition mechanisms between IPv6 and IPv4 need to be established. In Section 5 we examine ways to forward multicast IPv6-sessions through IPv4 infrastructure. We expect multicast-traffic to increase significantly when the benefits of IPv6-capability reach end users. Therefore the issue of proxying multicast traffic should be addressed immediately in order to facilitate the average user's ability to participate in IPv6 multicasts.

In this thesis a lightweight Linux-based dual-stack IPv4/IPv6 multicast tunneling proxy is built and tested. The goal is to successfully tunnel traffic for an experimental IPv6 application. Our ambition is to show that IPv6 multicast proxying is a viable solution for consumer-oriented products.

Limitations of our thesis include that tests have been conducted in a virtualized local network. Furthermore we have limited our tests to lightweight open source software implementations, excluding proprietary solutions such as products from Cisco or Juniper Networks. We have tested solutions for tunneling IPv6 over an IPv4 network, not translating IPv6 multicast to IPv4 multicast, which would be another solution [VENAAS].

In Section 2, a quick overview of current applications of multicast technology is presented. In Section 3 we provide an introduction to IPv6 from a tunneling and routing perspective. We then conclude the theoretical part by evaluating previous research in Section 4. We document our own implementations of IPv6 multicast proxying in Section 5. The

corresponding results are presented and analyzed in Section 6, together with suggestions on how to capitalize on the technology. Finally, in Section 7 conclusions and suggestions for future work are presented.

2. Applications of IPv6 multicast technology

As a consequence of the advantages of IPv6 (to be presented in Section 3) we believe that multicast will be more prominent in the future. This should facilitate a wide range of new group applications. Examples of potential applications for multicast are discussed in [RFC3170], we believe the most promising are:

IPTV Solutions An example of a one-to-many application is IPTV. To efficiently provide the bandwidth required for IPTV it is desirable to replicate the packets as far along the route towards the different hosts as possible (i.e., to take advantage of multicasting). Specific source multicast solutions for IPTV are very common and well researched, since it is one of the few applications today offering significant potential monetary savings for ISPs. This is particularly true for carrying traditional broadcast TV programming via IPTV - as there are typically a large number of subscribers looking at a limited number of different programs at a given time.

Synchronized Shared Resources A use-case for many-to-many multicast is synchronization of shared resources such as replicated databases. Using a single multicast session instead of several unicast streams saves bandwidth. Multicast communication also reduces the risk for deadlocks in replicated databases [HOLIDAY].

Other potential multicast applications include push services for software updates, distribution of stock prices, and other specific source data with many subscribers. Utilizing a multicast feed for software-updates, could potentially save bandwidth and reduce the time required to update a large number of computers. Push services could become more common thanks to the simplified allocation of group identifiers (GIDs). This means multicast could be widely used in webcam-conversations, net-meetings, etc.

The main application of multicast today, IPTV, has not been associated with a need for mobility. Therefore the problem of routing and proxying multicast packets has been overlooked until recently [RFC5757].[1] However, future services such as videoconferencing need to support roaming to be competitive with their unicast competitors. Therefore we believe that solving the problem of routing multicast packets is a requirement for successful multicast applications to emerge in the future.

[1] Some exceptions exist, such as [JIANG1] and [JIANG2].

3. An introduction to IPv6

To gain understanding of our implementation, we begin with a review of IPv6. IPv6 was first defined in 1996 to help solve some of the problems originating from the design of IPv4. The main problem is address exhaustion due to tremendous growth of the internet. It was clear that the IPv4 address space was not going to last much longer, as the number of devices connected to the internet was constantly growing [RFC2460].

IPv6 solves this problem by providing an address space that is much larger (with 128 bits of address space versus 32 bits of address space for IPv4). IPv6 also has a simpler header structure (see Figure 1) providing only minimal information. This makes it easier to parse, as most nodes along the route only need to know the destination address in order to forward the packet. Header extensions can be added to provide extra information when needed.

	Bits				
Bit offset from the start of the header	0-3	4-11	12-15	16-23	24-31
+0	Version	Traffic class	Flow label		
+32	Payload Length			Next Header	Hop limit
+64	Source address, 128 bits				
+192	destination address, 128 bits				

Figure 1: IPv6 header

There are two main methods of using IPv4 and IPv6 simultaneously. The first option is to use a dual stack network interface, so that the host or router can handle both IPv4 and IPv6 packets. This enables transparent usage of IPv4 and IPv6. IPv4 addresses are mapped to a special subset of IPv6 addresses by setting the first 80 bits to zero, the next 16 bits to one, and the IPv4 address itself is used as the last 32 bits.

The second option is to use tunneling, using a virtual device for the IPv4 or IPv6 interface, but only using IPv4 **or** IPv6 on the local link. Tunneling protocols are explained in Section 3.2

3.1. ICMPv6

ICMPv6 is the successor of ICMP for IPv6. It is used to transmit error and informational messages [RFC4443]. The format of an ICMPv6 packet is shown in Figure 2. The type field specifies the type of the message, the code field defines the message depending on the type and the actual message is in the message body.

	Bits		
Bit offset from the start of the header	0-7	8-15	16-31
+0	Type	Code	Checksum
+32	Message body		

Figure 2: ICMPv6 packet

Neighbor Discovery Protocol (NDP) is a subset of ICMPv6. NDP is used in IPv6 networks instead of ARP to determine the link-level addresses of nodes, and for finding routers that can forward packets [RFC4861]. In autoconfiguration a link-local IPv6 address

can be constructed by adding the 8 least significant bytes of the host's 64-bit MAC address[2] to fe80::0 [RFC4291]. When an IPv6 node is connected to a network it sends a link-local multicast router solicitation request to elicit a router advertisement from a router. The router responds with a router advertisement message that includes details of its network-layer configuration. As a result of IPv6's autoconfiguration and NDP, DHCP is not needed in IPv6 environments [RFC4862]. However, DHCPv6 can be used – however details of this lie outside the scope of this thesis.

When a node needs a link-layer address of a neighbor it sends a neighbor solicitation, and the node responds with a neighbor advertisement. When sending a Neighbor solicitation message the message is sent to the solicited-node multicast address, this multicast reduces the load on the network. The solicited-node multicast address can be formed by appending the 24 lowest bits of the host's IPv6 unicast address to FF02:0:0:0:0:1:FF00::/104 [RFC4291].

3.2. Tunneling protocols

Tunneling protocols can be used to encapsulate IPv6 packets in IPv4 packets so that IPv6 packets can be carried over a network which has **only** IPv4 support. In this section we will describe the most widely used protocols for tunneling IPv6 packets.

3.2.1. 6in4

One of these tunneling protocols is called 6in4. The IP protocol number 41 (IPv6-in-IPv4) is set in the IPv4 packet's header and the IPv6 packet is carried as the packet's payload. This encapsulation adds only a very small overhead of 20 bytes to each packet [RFC4213]. The protocol uses statically configured endpoints for the tunnel, but it is possible to use AICCU [AICCU] or a similar tool to get information from a Tunnel Information and Control protocol [TIC] server, that provides suitable tunnel endpoints. Figure 3 shows a semantic illustration of 6in4 tunneling.

Figure 3: Semantic illustration of the 6in4 tunneling protocol (with the lower red line indicating the tunnel and the black lines indicating the actual network links)

3.2.2. AYIYA

Anything-in-Anything (AYIYA) [AYIYA] can be used when a NAT prohibits protocol 41 traffic (protocol 41 as noted above is used for 6in4). AYIYA can also provide additional security by providing authentication of tunneled packets. AYIYA is very versatile and can use almost any protocol as the host or payload; because the packets are tunneled their payloads can be almost any higher layer protocol. In our case IPv6 packets would be the payload and UDP, TCP, or SCTP would be used as the host protocol. Figure 4 shows an AYIYA tunnel.

[2] The 48-bit MAC can be translated to a 64-bit EUI by adding FF:FE in the middle (i.e. a:b:c:d:e:f becomes a:b:c:FF:FE:d:e:f) [RFC4291].

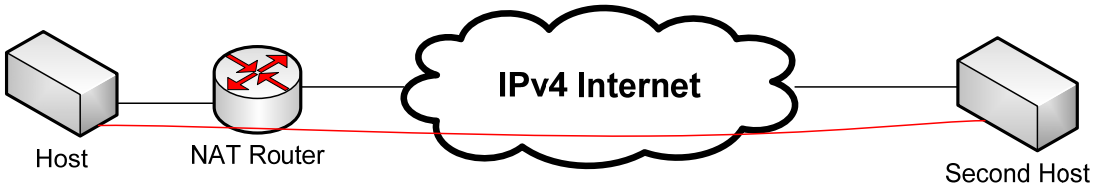

Figure 4: Semantic illustration of the AYIYA tunneling protocol (with the lower red line indicating the tunnel and the black lines indicating the actual network links)

3.2.3. 6to4

A third type of tunneling is 6to4 which is specified in [RFC3056]. This type of tunneling of IPv6 traffic avoids the need to configure explicit tunnels. All global IPv4 addresses are mapped into IPv6 address space. As shown in Figure 5 the high order 16 bits have the value 2002_{16}. The IPv4 address is placed in bits 16 to 47. The rest of the IPv6 address can be set to any value chosen by the user. Thus each IPv4 address user has control of a /48 IPv6 subnet.

bits	0-15	16-47	48-127
	2002	IPv4 address	User specified

Figure 5: 6to4 IPv6 address format

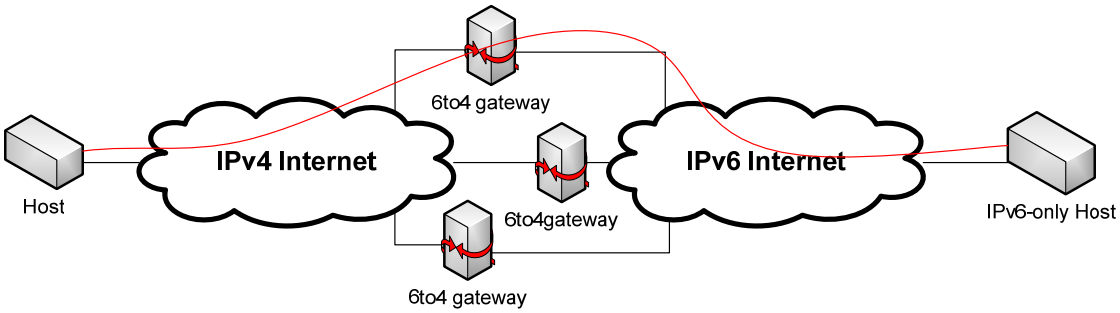

Figure 6: Semantic illustration of the 6to4 tunneling protocol (with the upperr red line indicating the tunnel and the black lines indicating the actual network links)

The outgoing packets from a node attached to an IPv4 only network are encapsulated using 6in4 and are forwarded to the destination IP address 192.88.99.1. This is an IPv4 anycast address for 6to4 gateways. The 6to4 gateway strips off the encapsulation and forwards the payload to an IPv6 internet. Incoming packets are encapsulated by the gateway, and forwarded to the customer's IPv4 address.

3.2.4. 6rd

IPv6 Rapid Deployment (6rd) is a new transition mechanism similar to 6to4. It was first used by the French ISP "Free", in the world's first large-scale rollout of IPv6 to end-users [RFC5569]. The standard is currently an Internet draft and presented in [6RD]. It was developed to solve the limitation of 6to4, in which it cannot be guaranteed that all native IPv6-hosts can reach all 6to4 hosts. The main difference between 6rd and 6to4 is that 6rd is bound to a single ISP. It does not use the 2002:: prefix for the IPv6 address, but rather uses IPv6 addresses from the ISP's IPv6-range. This is similar to the unicast IPv4 address being provided by the ISP.

3.2.5. Teredo

Teredo uses similar concepts to those discussed above for 6to4, but it does not need a public IPv4 address [RFC4380], so it works behind a NAT-device by using UDP datagrams (as these can pass through NATs) rather than using 6in4 encapsulation. Note that these UDP packets have to be permitted through the NAT if it implements any firewall functions. The Teredo server has a well-known address, and this server assigns the host a Teredo relay, IPv6 address, and punches a hole in the client's NAT. The Teredo relay is then responsible for the forwarding of encapsulated packets. The address format is specified in Figure 7.

bits	0-31	32-63	64-79	80-95	96-127
	2001:0000	Server IPv4 address	Flags	UDP port	Client IPv4 address

Figure 7: Teredo address format

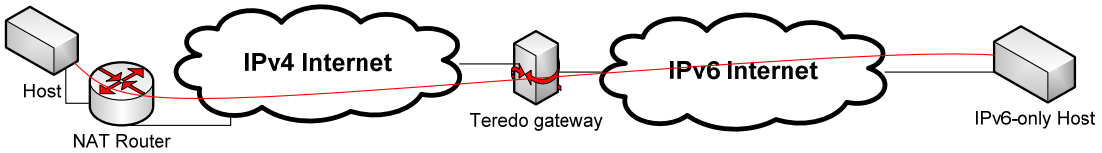

Figure 8: Semantic illustration of the Teredo tunneling protocol (with the lower red line indicating the tunnel and the black lines indicating the actual network links)

3.2.6. GRE

GRE is a tunneling protocol developed by Cisco. It is defined in [RFC2784]. It is similar to IP-in-IP-tunneling, but can carry almost all network protocols. The tunnel can also be used simultaneously for IPv4-traffic, resulting in a complete dual stack implementation that can handle both IPv6 and IPv4 multicast traffic. The GRE packet header has the form shown in Figure 9. Details of configuring a GRE-tunnel under Linux are provided in Section 5.3.3.

		Bits	
0	1-12	13-15	16-31
C	Reserved	Version	Protocol
Checksum (optional)			Reserved

Figure 9: GRE packet Header

Checksum Present (bit 0): Set to one if the Checksum is present, 0 otherwise.

3.3. The IPv6 multicast address range

As in IPv4, a multicast-address does not specify a specific node. Instead an IPv6 multicast address serves as an identifier for a group of nodes that form a logical group (hence a multicast message is sent to such a group address). The format for multicast addresses is defined in [RFC4291] and shown in Figure 10.

8 bits	4 bits	4 bits	112 bits
Multicast	Flags	Scope	Group ID

Figure 10: IPv6 multicast address (adapted from [RFC4291])

Multicast field: The first 8 bits indicate a multicast address by being set to all ones, i.e., FF_{16}.

Flag field: The left-most flag bit is indicated as reserved in [RFC4291], this means that it should be initialized to zero. The low order (right most) flag-bit should be set to "1" if the address is transient (not permanent). Only well-known addresses that have been assigned by an authority such as IANA may have this flag bit set to zero, indicating that this is a permanent address[3]. The middle two bits are discussed later in this section.

Scope field: The scope field specifies the scope in which the address is valid (for example: global or site local). There are some fixed scope permanent addresses, such as the "all nodes address" that is always link or node local. However, fixed group IDs are valid in all scope settings. For example, the UPnP-group 130 can be a link local or global address depending on the scope specified in the scope field. The defined values for scopes are maintained by IANA and the current values can be found in [RFC4291].

Group ID field: Most multicast IPv6-packets will be encapsulated in Ethernet-packets with their MAC addresses based on the last four octets of the IPv6 multicast address. The resulting Ethernet MAC address has the format: 33-33-w-x-y-z. Therefore, it is recommended to only assign group IDs from the final four octets of the 112-bit range. Doing so implies that all group IDs will be associated a unique Ethernet MAC-address [RFC2464]. The rest of the group-ID field should be zero. Group IDs have been divided into three blocks: permanent addresses (0x00000001 to 0x3FFFFFFF), permanent group identifiers (0x40000000 to 0x7FFFFFFF), and dynamic addresses (0x80000000 to 0xFFFFFFFF, where the transient bit is set to 1) [RFC3307]. The first two types of identifiers are assigned by the IANA based on expert review. In contrast dynamic addresses can be allocated spontaneously, via a range of protocols such as the Multicast Address Dynamic Client Allocation Protocol [RFC2730] or Zeroconf Multicast Address Allocation Protocol [ZMAAP]. The wild-west situation regarding dynamic assignments of group IDs has led to an extension of the initial address space definition for unicast prefix based IPv6 Multicast Addresses. Such an address is shown in Figure 11.

8 bits	4 bits	4 bits	8 bits	8 bits	64 bits	32 bits
Multicas	Flags	Scop	Reserved	Network prefix length	Network Prefix	Group

Figure 11: Unicast prefix based IPv6 Multicast Addresses (adopted from [RFC3306])

The unicast prefix based IPv6 Multicast Address extension is indicated by setting the 11^{th} bit. By masking a 64-bit network prefix we can convert a unicast network prefix into a multicast address-space. This enables network operators to be able to keep track of their multicast addresses, and administer in them a flexible manner. Furthermore, source-specific multicast addressing becomes possible. Source-specific multicast has been popular in the IPv4-infrastructure since it solves the problem of address collisions [RFC3569].

3.3.1. Multicast Listener Discovery (MLD)

To achieve multicast functionality (as opposed to broadcast functionality), routers must be aware of what groups that have nodes attached to each of its ports. The Internet group management protocol (IGMP) was engineered as an extension to IPv4 to enable nodes to join/leave multicast groups [RFC3376]. Multicast Listener Discovery (MLD) is a direct successor of IGMP, with very similar functionality. However, MLD messages are sent as ICMPv6-packets instead of IGMP-packets (as IGMP and ICMP were unified into ICMPv6). MLD consists of three message types: Query, Report, and Done [RFC2710].

[3] See http://www.iana.org/assignments/ipv6-multicast-addresses/.

On each link, only one router is normally allowed to send MLD queries, this router is designated the Querier. Upon initialization, every link on every router is a Querier until it receives a Query from a router with a smaller IP-address value.

When an upper layer application on a host requests the host to start listening to a multicast group, the host sends an unsolicited MLD Report over its interface with the group multicast address as the receiver. Routers that receive reports append the GID of this MLD Report to its list of GIDs with attached listeners, and the interface(s) involved. It will also relay the Report via other links if needed. Subsequently the router will periodically Query for listeners, and flush groups for which it does not receive Reports in response. Hosts that want to stop listening to a group may also send a Done-message. Before replying with a Report to a Query, all hosts wait a conditionally random amount of time. If they receive a Report before the time expires, they do nothing. Otherwise they broadcast a link-local report. This ensures that a link with a large number of listeners in a group is not flooded with Report-messages. This state machine is shown in Figure 12.

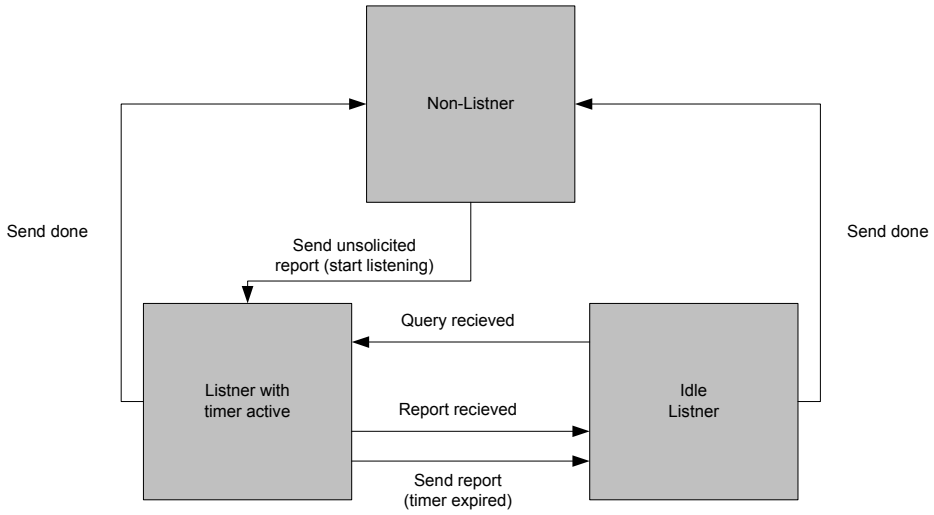

Figure 12: MLD state machine (adapted from [RFC2710])

The protocol ensures that all routers update their multicast routing tables. Multicast traffic received for groups without listeners is simply dropped. MLD poses several challenges for any node wanting to forward multicast traffic. The hop-limit of MLD-packets is always 1, to ensure that only directly attached routers receive Responds/Queries. While it might seem simple for the sender to simply set the MLD packet hop-limit to greater than 1, tampering with the hop-limits could yield severe loops. As a compliment to the host to router communication provided by MLD, separate multicast routing protocols are needed for inter-router traffic [RFC2710].

3.3.2. Routing multicast packets

When a router receives a packet, it needs to know the best route to the destination of the packet. Two main routing-strategies are used to handle multicast-traffic: Source Based Trees and Group Shared Trees. One could say that they are opposites of each other, since the trees are built “host to source” in the former, and “source to host” in the latter.

Source based trees are shortest path trees (SPTs) rooted at multicast routers. They need to be generated individually for every router in a network. Furthermore a separate tree is needed for every multicast group, since each multicast group has its own unique member-

topology. If a network has *n* routers and *m* active mulicast groups, then a network total of *nm* source based trees need to be generated. This implies a lot of control-traffic between routers. The use of source based trees is therefore optimal in environments featuring a high concentration of multicast clients.

Link state routing, which is often used to create source specific trees in unicast, can easily be expanded to support multicast, as in the MOSPF [RFC1584] protocol, a data driven protocol where SPTs are calculated on-the-fly and cached. Disadvantages include large amounts of control traffic and the extra time required for route calculation.

Distance vector routing is a simpler approach to SPTs in uncast, but its extension to multicast is non-trivial. A primitive solution is Reverse Path Forwarding (RPF). When a router receives a multicast packet, it does a reverse check in the routing table, to determine if the packet came from the shortest path from the source. If so, it forwards the packet on its other interfaces, otherwise the packet is dropped. RPF creates a lot of unnecessary traffic, and furthermore is prone to producing duplicate packets. Therefore, together with protocols such as MLD, an improvement called Reverse Path Multicast Routing is possible. Duplicate packets are avoided by assigning a designated parent router in each network, and excessive traffic is avoided thanks to the Report/Done/Query messages in MLD. Implementations of Multicast Distance Vector Routing include DVMRP [RFC1075] and PIM-DM [RFC3973].

In reality, many multicast networks are not dense enough to justify the overhead of generating source based trees for every router. Using group shared trees as a method for routing multicast packets has become increasingly popular. Instead of creating a SPT in every router, a single router is designated a rendezvous-point for the autonomous system (AS). Information about this router is broadcasted to other routers and hosts within the AS. When a host wishes to join a multicast group it sends a join message to the rendezvous-point. This message is registered by every router on the way to the rendezvous-point. After a certain period of time, a spanning tree is built from the rendezvous-point, thus all nodes will be able to communicate via the rendezvous-point. An implementation of group shared trees is Core Based Tree (CBT) [RFC2201]. One of the challenges in group shared trees is the issue of selecting the router to be the rendezvous-point, CBT solves this via a designated “HELLO”-protocol [RFC2189]. PIM-SM is a popular and more widely used group shared tree protocol implementation [RFC4601].

Yet another routing protocol that is relevant for IPTV is PIM-SSM [RFC3569]. Unlike PIM-SM which supports any-source multicast within a group, PIM-SSM only supports unidirectional data sent from a specific source. A SPT rooted at the router closest to the source is constructed via PIM-SM and any host can listen to different “channels” by taking both source unicast address and group ID into consideration.

A problem with PIM-SM, the most widely used multicast routing protocol, is that inter-AS communication is not supported. A consequence is that any source multicast between two multicast domains becomes problematic since Rendezvous Points (RPs) cannot exchange information about connected nodes. A potential solution has been presented in [RFC3956]. The authors suggest expanding the flag field of the IPv6 multicast address (see Section 3.3) to include an “R-bit” and using four of the previously reserved bits to store an RP Interface ID (RIID). The RP address can be constructed by taking *plen* bits of the network prefix, inserting zeros until the RIID (last 4 bytes). This would make it possible to embed the RP-address in every multicast packet, meaning that inter-domain any source multicast would become possible in IPv6.

4. Previous work

Two KTH students have built a fully functional IPv4 multicast proxy [NILSSON]. Their solution was based on an application called IGMP Proxy, which forwards multicast-traffic while handling the relay of IGMP-messages. For benchmarking they used VLC to stream video. A group of students at the Hochschule für Technik Rapperswil[4] published a similar project report regarding an IPv4 based "multicast proxy server" in 2001 [GLANZMANN]. Based on these we have identified three main issues for the success of IPv6 multicast namely proxying or interconnecting IPv4-space with IPv6-space, forwarding MLD-packets, and routing IPv6 multicast packets.

Challenges related to interconnecting multicast nodes in IPv4-space with other nodes in IPv6-space have been addressed by several papers. A recent draft from Huawei Technologies presents a solution in the form of a translating multicast proxy to be placed in the border between IPv6-space and IPv4-space [JIANG3]. A similar idea was presented at a conference in 2007 [BLAGA].

The issue of forwarding MLD packets in an appropriate manner has been discussed in several publications. A group of researchers from Motorola published a draft for IPv6 multicast forwarding to mobile nodes, also implementing a MLD-proxy in 2004 [JANNETEAU].

Regarding IPv6-functionality, a bachelor's thesis by Thor Hådén demonstrated how to build a functional IPv6-router with no/minimal customization of a Linux 2.6 kernel [HÅDEN]. Instead of building a proxy, he built a router to give IPv6-functionality to radio-based accessories that might be used in a home automation system. His reason for using IPv6 was to enable the potentially large numbers of these devices to be directly addressed and controlled from anywhere in the Internet.

There are several existing software implementations capable of forwarding multicast IPv6-packets. Therefore, it should be possible to set up a proxy that is much more advanced than the two previous KTH-projects. Hardware implementations such as MLD-proxy (which works in a similar manner to ecmh) from Juniper Networks is also a solution for IPv6 multicast forwarding [JUNIPER]. The Linux 2.6.x kernel has offered experimental support for multicast forwarding since 2008. However, this requires additional knowledge and configuration. This additional information will be presented in the next section.

[4] Part of the University of Applied Sciences of Eastern Switzerland

5. Implementation

5.1. Goals

Few papers have dealt with the holistic challenge of building an IPv6 multicast proxy. This will be the focus of this section. Different tunneling-protocols will be examined and tested, along with three different proxy architectures. Our ambition is to build a Linux based dual stack IPv4/IPv6 multicast proxy and benchmark it with IPv6 applications.

5.2. Testing methodology

After some preliminary research and deliberation we decided not to benchmark the performance of the different proxy-platforms or tunnels in terms of throughput, memory usage, or CPU usage. Because we use virtual machines, it is challenging to benchmark the performance of the virtual guests independently of each-other and the physical host [DRUMMONDS]. Instead we measure and analyze the latency of the different platforms. Thus, the metrics we will focus on are usability, flexibility, ease of implementation, and performance (in terms of latency).

5.3. Testing environment

After careful research and deliberation, a decision was made to implement a fully virtualized testing environment. Virtualization provides several advantages over physical testing-environments. The most obvious advantage is cost-efficiency, since less hardware and electrical power is required for the implementation. Furthermore the setup is simplified, since virtual machines need to be installed only once. A set of machines can be replicated by simply copying the first machine's hard-drive image and customizing each virtual machine's configuration files. Finally thanks to scripting and monitoring the environment via packet sniffers data-collection can be done very effectively. These benefits of using virtual machines, scripting, and collecting packet traces for subsequent analysis enabled us to conduct multiple test runs and explore several alternative configurations and approaches.

Our virtual environment consists of all the virtual machines needed to deploy and benchmark an IPv6 multicast network. The following four virtual machines (shown in Figure 13) run concurrently:

Router	A dual-stack IPv6/IPv4 multicast capable router that serves as the central node of the test environment.
ISP	Sends multicast-packets to a specific multicast group on its physical interface.
Home	This is our proxy. It receives multicast-packets on its physical interface, and forwards them though a unicast (but multicast capable) virtual tunnel interface to the away-machine.
Away	Receives the multicast packets forwarded by Home.

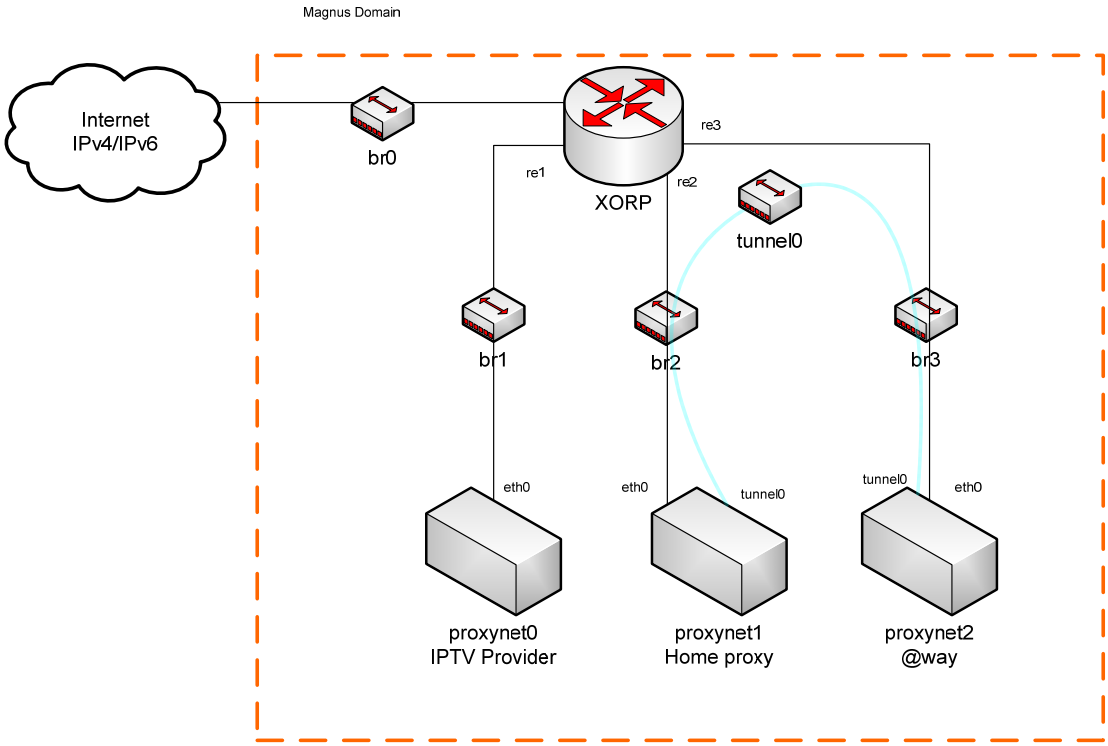

Figure 13: Virtual network diagram

Before constructing the proxy we needed to construct a virtual dual-stack IPv4/IPv6 router with multicast forwarding capabilities. The Linux kernel could potentially be configured for this, but we decided not to. The Linux kernel IPv6 multicast forwarding documentation is rather unrefined at the moment, meaning that a faulty router could be a source of errors. Instead we used the eXtensible Open Router Platform (XORP) as our platform (this software is running on a virtual machine running FreeBSD - see Section 5.3.2).

We chose to use Ubuntu Linux 9.10 Server Edition as the platform for the virtual machines. However, we compiled a customized kernel for the proxy machine in order to enable IPv6 multicast forwarding and PIM2 (see Section 5.6) for this particular machine. The remaining two virtual machines ran the Ubuntu 9.10 Server Edition with a vanilla kernel.

To enable the rapid deployment of several different types of proxies, a simple method for testing the functionality of the proxies was needed. We decided to use a simple python-script for this purpose. The script can be run in server or client mode. In server-mode, the script joins a multicast group and generates a stream of multicast-packets. In client-mode the script joins the corresponding groups, and generates screen output when multicast packets are received. By running this script in server mode on a virtual ISP-machine, and running the script in client-mode on the virtual away-machine (a client machine that is located away from the user's home network) we were quickly able to confirm the functionality of the different proxy-implementations. This script is described in more detail in Appendix A.

5.3.1. Configuring KVM

The main factors when choosing the hypervisor for our virtual environment were performance and flexibility. The Kernel Virtual Machine (KVM) is a unique hypervisor in the sense that it is integrated into the Linux Kernel, yielding good performance [KVM]. KVM also features a QEMU front-end offering good opportunities for customization [QEMU]. Even though four virtual machines were run simultaneously, performance on our modest hardware (details in appendix F) was quite acceptable.

In order to interconnect the virtual machines on the data-link level virtual Ethernet (TAP) devices were created using the Universal TUN/TAP-drivers for Linux. Each virtual network-card was associated with a TAP-device via the QEMU front-end. The TAP-devices were then interconnected via virtual bridges using the Linux bridge utilities. An overview of the final network is given in Figure 13.

We decided to use QEMU's command line parameters as the means for initiating, setting up, and tearing down the virtual environment via a shell script. This is a simple and sufficient solution, since the configuration of the virtual machines does not need to be changed dynamically during run-time. The script starts by setting up three bridges, and seven TAP-devices. It then starts the four virtual machines and interconnects them via the already created network devices. An alternative to using a shell script would be to use python with the libvirt API. Although this would add complexity to the implementation, it would make the script portable to other virtualization platforms such as Xen with minimal reconfiguration. The final shell-script for setting up our environment is included in Appendix C.

5.3.2. Configuring XORP

XORP (eXtensible Open Router Platform) is an open-source routing software suite available for a wide range of platforms including Windows and Linux [XORP]. The software supports most commonly used protocols and has a modular design that makes implementations of new features quite straightforward. In order to maximize compatibility and minimize the amount of configuration needed we decided to use the Live-CD version of XORP. This version runs on top of a FreeBSD operating system. This gave us the opportunity to experience a new operating system environment.

The process of configuring a XORP router is very similar to the process of configuring Cisco or Juniper hardware. The configuration can either be loaded from a USB-drive during start-up or edited at run-time using the command line utility XORP-shell. Features were enabled and tested in the following order: IPv4 unicast, IPv4 multicast, IPv6 unicast, and IPv6 multicast.

In our testing environment, the network was divided into three subnets, one for each virtual machine. For the sake of simplicity static routing was used for unicast packets. The process of configuring unicast IPv4 and unicast IPv6 was very simple, and gave us a good introduction to the XORP shell configuration utility. For multicast routing, we decided to use dynamic routing in the form of PIM-SM. Configuring multicast was challenging, mainly because of confusing debug messages from XORP while setting up PIM-SM. The most important aspect of the PIM-SM configuration is the fact that the router must be configured as a candidate RP (rendezvous point, see Section 3.2.2) since no other router will be present in our test network. After solving this, the router worked flawlessly for both unicast and multicast traffic for both IPv4 and IPv6. The complete configuration file is discussed in Appendix D.

The complete configuration file was loaded onto a USB-drive that was mounted by the virtual router upon initialization. The longer boot time due to using a live CD in combination with a read-only USB-drive for the configuration was tolerable because data cannot be lost by a corrupt hard-drive image. During this project XORP proved to be very reliable. Our opinion is that XORP should be considered a viable substitute to products from Cisco or Juniper for small-scale networks.

5.3.3. Configuring Tunnels

Initially we chose to use a 6in4-tunnel for our implementation, as it is the easiest tunnel approach to implement in a Linux environment. The tunnel was configured statically in the /etc/network/interfaces files on both virtual machines. As it is a stateless tunneling protocol it does not need any other configuration to function properly.

During testing we had to abandon the 6in4 tunnel and replace it with a GRE-tunnel to in order to get kernel multicast routing to work. The GRE tunnel was configured with the ip(8) tool under Linux. When specifying a GRE-tunnel with the ip(8) tool in Linux the kernel will not compute a checksum for the packets unless the argument "[i|o]csum" is passed to the tool. The kernel uses protocol version number 0 as specified in [RFC2784]. The configuration files can be found in appendix E.

5.4. Python Script forwarder.py

The first proxy implementation tested is the python script "forwarder.py", which is based on the script we used to test our setups. The basic script is provided by the python project as an example of using multicast support. We modified it to support both receiving and sending multicast packets at the same time. This means that the script acts as a multicast-relay. The script has to be configured and set up in the same way as the "mcast.py" script we used for testing purposes. Our main reason for including this python script based implementation is to show how easy it is to implement IPv6 multicast features with python. The complete script can be found in appendix B

5.5. ECMH

The second implementation for testing our proxy was Easy Cast du Multi Hub (ECMH). This tool is similar to IGMP Proxy, but has support for IPv6. The software is based on [RFC4605], and was written by Jeroen Massar in 2004. It runs on most POSIX-compatible platforms, but was most widely used on the Linux operating system before the Linux kernel included support for multicast routing. It does not support routing with PIM, but like a normal multicast router, it listens to MLD reports from all interfaces and sends MLD queries.

The Figure 14 shows a typical use-case for ECMH, where it automatically listens to MLD messages from hosts A, B, and C and forwards them to the PIM router. The PIM router then forwards the multicast messages to the hosts wanting to receive them. The ecmh router does not need any configuration, as it starts listening to all interfaces for MLD packets upon initialization. The upstream interface can be configured by starting ecmh with the -i 'interface' parameter.

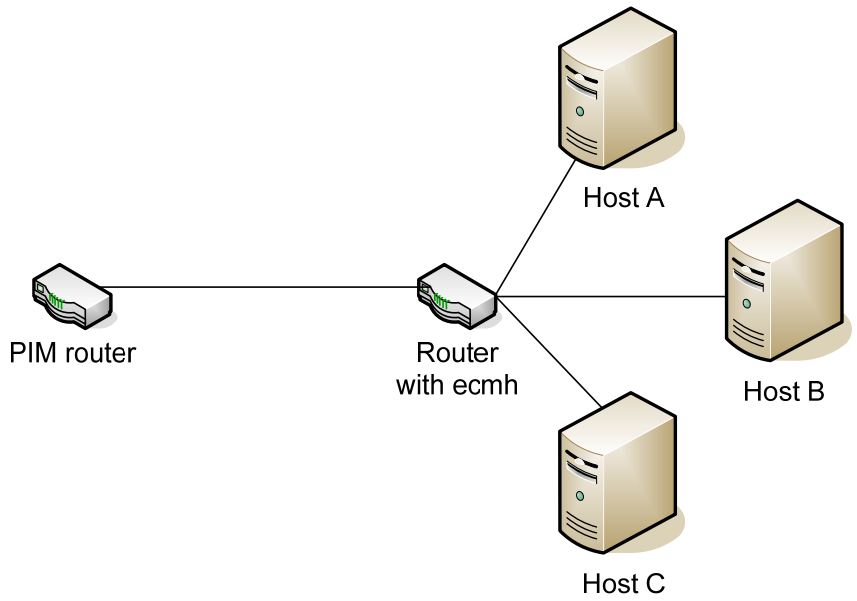

Figure 14: ECMH - adopted from the m6bone mailing list message by Stig Venaas on 2004.9.19

To test ecmh we downloaded the ecmh Debian package from http://unfix.org/projects/ecmh/ and installed it on the proxy. We started the software with the command "sudo ecmh -i eth0" as well as starting the "mcast.py" scripts first on the server and then on the receiver. To get ecmh to work, we had to give the tunnel endpoints IPv6 link-local addresses. Ecmh did not co-operate fully with the multicast forwarding enabled kernel. It did not forward packets unless the kernel multicast was enabled (e.g. by starting up smcroute). We are not sure why this happened. Further research is suggested, as if the Linux kernel includes multicast forwarding by default in the future, then ecmh would not work as expected.

To get information from a running ecmh instance a SIGUSR1-signal can be sent to it with using the "kill –USR1 *PID*" command, where PID corresponds to the ecmh process ID. It then outputs the statistics to /var/run/ecmh.dump, where they can be read. An example is provided in appendix H.

5.6. Linux Kernel + smcroute

Support for IPv6 multicast forwarding in the Linux kernel was developed by Mickaël Hoerdt during 2004. It was distributed as a separate patch for the Linux 2.6.9 kernel until 2005, when the code was merged into the USAGI (Universal playground for IPv6) project. USAGI is a Japan-led project to implement stable IPv6-functionality for Linux [LINUX-IPV6]. The project is sponsored by several Japanese universities and corporations, and has contributed almost all of the Linux IPv6 kernel code. USAGI is currently implementing Mobile IPv6 for Linux. In late 2008, the USAGI contributor Hideaki Yoshifuji committed experimental IPv6 multicast forwarding and PIMv2-functionality to the mainline Linux 2.6 kernel. As a consequence of its experimental and potentially unstable nature, the IPv6 multicast forward module and PIM-module are not included in most distributions (including Ubuntu). The solution is to download and compile the latest Linux kernel source-code, and edit the kernel configuration file to include following parameters:

CONFIG_IPV6_MROUTE=y
CONFIG_IPV6_SUBTREES=y
COMFIG_IP_PIMSM_V2=y

In our case, we decided to download the latest Ubuntu kernel source from their git-repository and branch it into our own flavor, which we called 2.6.31-20-ipv6. We edited the relevant configuration lines in the networking section of the configuration file, where they were previously commented out. After compiling and installing the new source, the fresh kernel offered some interesting new options and attributes. The conventional method of viewing and changing kernel parameters during run-time has been to use the utility sysctl(8), but this utility is being depreciated. Instead the memory-based /proc/ file-system is used to change system parameters. The most important parameters are summarized in Table 1.

Table 1: Run-time parameters for IPv6 multicast routing in the Linux kernel

Parameter path	Description
/proc/ip6_mr_cache	Gives the state of the kernel's multicast forwarding information base
/proc/ip6_mr_vif	Lists all interfaces supporting multicast routing.
/proc/sys/net/ipv6/conf/ *interfacename*/forwarding	Show the status of multicast forwarding for the interface named "interfacename".

In the initial kernel-patch by Hoerdt these parameters were editable in user-space. However, since the code was committed to the mainline kernel, these parameters need to be edited using a kernel-level application. Luckily several applications are available, such as smcroute (static routes), mrouted (DVMRP), and PIMd (PIM-SM). Since our proxy only needs to handle a static route where all traffic is sent to a tunnel-interface the functionality in smcroute is sufficient. The traffic can be directed to its final destination by changing the tunnel end-point.

We downloaded and installed the latest[5] smcroute .deb package available from the Ubuntu repositories instead of compiling the application from source. The program runs as a daemon. Static routes are added by calling:

smcroute –a <input interface> <original sender ip> <multicast group id> <outputinterface>

in our case

smcroute –a eth0 fc00::2 ff15:7079:7468:6f6e:6465:6d6f:6d63:6173 tunnel0

An issue is that applications running wherever tunnel0 leads to (in our case *away*) have no way to send "join"-messages unless the MLD-messages can be proxied. Therefore smcroute includes the option to send an arbitrary "join"-message through the input-interface. This is is done via the command:

smcroute –j <inputinterface> <multicastgroup>

in our case

[5] Version 0.94.1

smcroute –j eth0 ff15:7079:7468:6f6e:6465:6d6f:6d63:6173

After setting up the environment we tried sending some test-packets from a virtual machine (fc00::2, our python server). We can use cat(8) to read /proc/net/ip6_mc_vif in the proxy to see the number of multicast packets being sent via the interfaces and tunnels (note that all of the packets have been forward to the tunnel):

```
elis@proxynet1:~$ cat /proc/net/ip6_mr_vif
Interface      BytesIn  PktsIn  BytesOut PktsOut Flags
 0 eth0           2066      31         0       0 00000
 1 tunnel0           0       0      2066      31 00000
```

Similarly, /proc/net/ip6_mc_cache reveals:

```
elis@proxynet1:~$ cat /proc/net/ip6_mr_cache
Group                               Origin
Iif       Pkts   Bytes    Wrong  Oifs
ff15:7079:7468:6f6e:6465:6d6f:6d63:6173
fc00:0000:0000:0000:0000:0000:0000:0002
0         31     2066     0      1:1
```

We can see that both interfaces have been initiated in the kernel as virtual interfaces for use in the multicast routing database, and the correct static route has been set up. Furthermore the same number of multicast-packets have passed though both interfaces.

The packets were received by our python-client at the other end of the tunnel. This means that the proxy worked as expected. The process of setting up IPv6 multicast forwarding in Linux is therefore quite straightforward. However, because of the experimental nature of the kernel modifications, and the lack of documentation for smcroute, the implementation did not work as expected. Initially when initializing smcroute, none of our tunnel devices were registered under the /proc/net/ip6_mr_vif. After a large number of trials with different tunnels and setups, and some e-mail support from the author of the application, we finally managed to register a tunnel interface. We only managed to register GRE-tunnels, after explicitly setting up their multicast-support before initializing them. It seems that the tunnels are not recognized as proper IPv6 multicast devices during the initialization of smcroute. The following output shows that only the “all”-interface, our physical interface, and one of our tunnels are activated for IPv6 multicast forwarding (indicated with a value of 1). The inactive interfaces are non-GRE tunnel devices (indicated with a value of 0).

```
elis@proxynet1:~$ grep –G [*]*
/proc/sys/net/ipv6/conf/*/mc_forwarding
/proc/sys/net/ipv6/conf/all/mc_forwarding:1
/proc/sys/net/ipv6/conf/default/mc_forwarding:0
/proc/sys/net/ipv6/conf/eth0/mc_forwarding:1
/proc/sys/net/ipv6/conf/gre0/mc_forwarding:0
/proc/sys/net/ipv6/conf/lo/mc_forwarding:0
/proc/sys/net/ipv6/conf/tunl0/mc_forwarding:0
/proc/sys/net/ipv6/conf/tunnel0/mc_forwarding:1
```

After successfully initializing a tunnel interface, the kernel forwarding worked quite well. Debugging was simple thanks to the parameters in the /proc/ file system. During our trials smcroute crashed and exited without warning several times, indicating that this code is not stable enough for business critical use. With some debugging of the code in the kernel and smcroute and the addition of some much-needed documentation, we believe that this kernel implementation is a viable solution. The code integrates seamlessly with the already functional IPv4 forwarding code in the kernel. Furthermore, the fact that user-level programs

use API-calls to the kernel instead of implementing their own code minimizes otherwise tedious code-repetition and centralizes the multicast route information base (MRIB). To facilitate others using this software we have contributed our document to the project's maintainer(s), including the bug report in Appendix G.

6. Results

6.1. Analysis of implementations

6.1.1. Flexibility

The main strength of ECMH is its ability to operate in many types of networks. The fact that it makes routing decisions based on MLD-packets implies it will work independent of any dynamic routing architecture, such as PIM. As a consequence, ECMH is very useful as a multicast routing solution in nodes that do not offer installation of new software, such as home gateways/routers.

Our python-script has static routes hard-coded into its source, making it rather useless outside our testing environment. However, it could easily be extended to add static-routes during run-time along with other desirable features. Because the script is written in python, the script should be platform independent.

Having the bulk of the implementation in the kernel of the operating system is an advantage of the Linux kernel implementation. The implementation enables the system to be flexible through a suite of user-level daemons, while minimizing code-repetition. By using daemons such as PIMd, mrouted, and smcroute, the system can support several different routing-protocols, while the bulk of the work is done by the kernel instead of in user-level code. Unfortunately, these programs are outdated and we only managed to install smcroute, thus limiting us to static multicast routes.

6.1.2. Ease of administration

ECMH needs no configuration, thus its administration is trivial. Additionally, it features an informative debug mode, which is helpful when trying to pinpoint errors in the system. The python-script cannot be modified at run-time which was a disadvantage. Furthermore, the script uses IPv6 addresses hard-coded into its source, which is not user-friendly. It does not feature any debug information, and therefore debugging has to be done with network sniffing tools such as Wireshark.

The smcroute daemon features a set of commands for changing routes and changing the router behavior during run-time. The opportunity to read and view the /proc/ file system for information regarding the kernel-parameters, combined with the verbose output option of smcroute enabled effective debugging of the system.

6.1.3. Ease of implementation

ECMH was the simplest solution to set up and benchmark. After installing the Debian package and initializing the program, the routing just started to work. No configuration file or command line parameters were needed. For simple multicast “routing” such as the type needed in home-proxies, this solution is the most appropriate of the ones we have tested. A limitation of ecmh is that it does not have any loop detection. If it is used in a home-proxy it could lead to a non-functioning network.

Setting up a python-script to forward multicast-traffic gave us a good introduction to multicast socket programming. The socket-API for python is very simple to use, and is backed by an informative and helpful community. The script took some time to write and

debug, thus the set-up process was lengthier than that for ecmh. However, this hands-on approach gave us a better understanding of the inner workings of the implementation.

Multicast forwarding in the Linux kernel is quite challenging to set up. It requires compiling a custom kernel, and the use of outdated user-level applications (even for static routes) in order to change the kernel-parameters at run-time. Furthermore, documentation for the kernel API is more or less non-existent. Instead, we have relied on the documentation for Mickaël Hoerdt's ancient patch that the kernel-code is based on. The documentation for smcroute is better, but we still needed direct support from the author to get our tunnels activated as interfaces in the MRIB. To remedy these shortfalls we have submitted a bug report (see appendix G).

6.1.4. Performance

By using Wireshark to measure the time difference between the entry and exit of packets from the proxy, we measure the latency of the routing process. An overview of our results based upon running each of the different routing processes three times and forwarding 100 multicast packets in each run is presented in Figure 15. A statistical analysis of these sets of data is shown in Table 2.

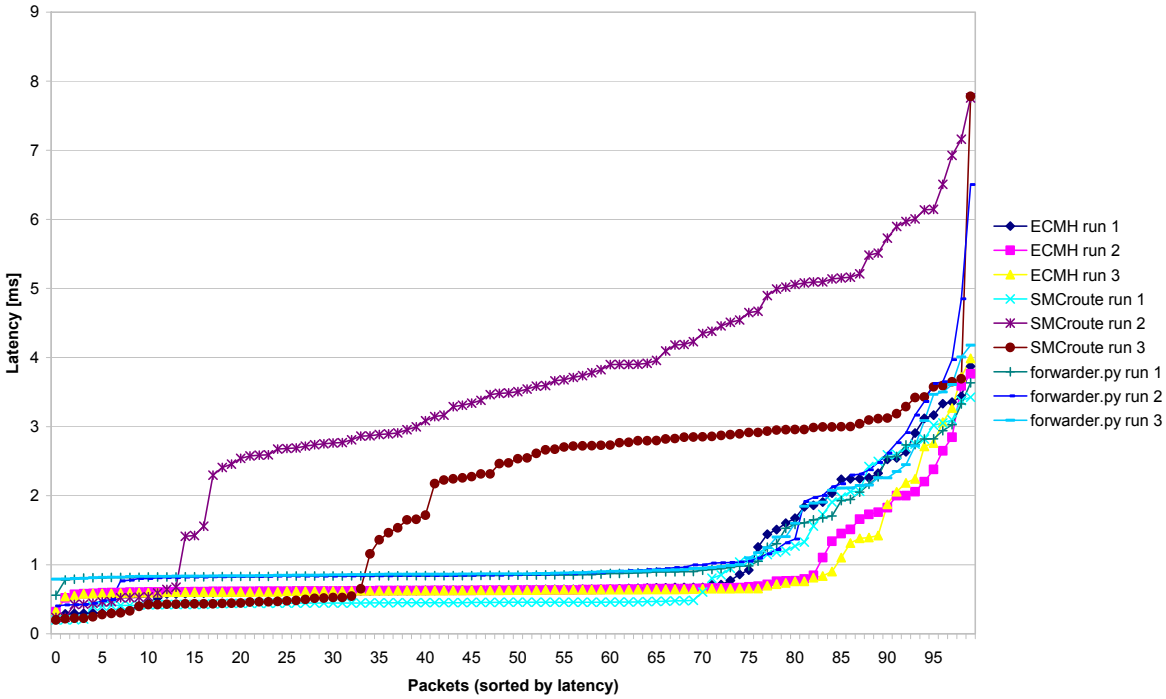

Figure 15: Internal latency in proxy implementations

Table 2: Statistical analysis of internal latency test data

Latency	**Forwarder.py**	**ECMH**	**SMCroute**
Average [ms]	1.22	0.93	2.11
Median [ms]	0.86	0.63	2.27
Variance [ms]	0.82	0.73	1.70

We were surprised to find that the worst implementation in terms of delay performance was the Linux kernel (shown in the latency performance in the column labeled

SMCroute in Figure 15). Both ECMH and our python script yielded a lower and more consistent latency. This was unexpected since we assumed that the kernel implementation would be the most efficient solution since the operating system needs fewer context-switches to process the packets. The bad performance leads us to question the quality of the code in the Linux multicast forwarding engine. We should note that this poor latency was true for two of the three runs using SMCrouter (specifically runs 2 and 3), but not true of the first run. We are not sure of the reason for this.

6.1.5. Comparison of implementations

In order to create a simple overview of the results of our comparison, we rate each implementation in a quantitative manner with respect to the categories defined in section 6.1 and the latency performance as described in section 6.1.4.

Table 3: Quantitative comparison of the proxy implementations (1 indicating the lowest rating and 3 indicating the highest rating)

	ECMH	Kernel	Script
Flexibility	3	2	1
Ease of use	3	2	1
Ease of implementation	2	1	2
Performance	3	1	2
Total	11	5	5

Our conclusion is that ECMH is the most holistically useful platform for forwarding packets in an IPv6 multicast proxy. It yields the best performance and is very simple to configure. Both of the other platforms potentially could be equally good (if not better) than ECMH, but because of unstable code, lack of documentation, and volatile latency performance these platforms are not ready to be used today.

6.2. Potential Capitalization Strategy

6.2.1. Existing best-practices

A potential use-case for a dual stack IPv4/IPv6 proxy would be remote access to an IPTV subscription. A user could use a proxy at home to route traffic from his or her IPTV-subscription to another device anywhere in the world. A similar concept for conventional television already exists in the United States. The solution is sold by Sling Media under the brand “Slingbox” [SLING]. The Slingbox re-encodes TV-signals into a digital format and proxies a stream to digital devices such as cell-phones and laptops. Issues such as NAT and dynamic IP addresses are resolved by Sling Media though a central server that tracks active Slingboxes, and their respective owners.

6.2.2. Suggested value proposition

We believe that the multicast technology reviewed in this thesis could provide the technological core for a similar product aimed at the IPTV-market. Many IPTV-providers are subsidiaries of conventional television-providers that feature geographically relevant content such as local news. Furthermore, television networks in the United States still hesitate to release high profile content outside the United States directly. Therefore the incentives that resulted in a demand for the Slingbox also exist in the IPTV business. If packaged and sold as a consumer oriented solution, our proxy would provide a simple method of accessing an IPTV

subscription from anywhere. Note that the user's proxy receives the traffic in the same local network that their usual IPTV set top box is located in.

The core activity of a company capitalizing on IPTV technology would be the construction, distribution, and marketing of IPTV proxies. However, we doubt that the product alone would be a unique selling point. Since the technology already exists in the public domain, the entry barriers for new competitors would be low.

Instead, the deployment and maintenance of servers for value-added features, software updates, and facilitated interconnection of the proxy to devices would be important auxiliary activities. An example of a value-added feature could be subscriptions to third-party media. Apart from adding extra features to the value-proposition, these servers would lock customers to a specific solution. Furthermore, additional revenues through a subscription model would be possible.

6.2.3. Target market

According to an industry expert group IPTV is projected to feature over 80 million subscribers worldwide by 2013 [IPTV]. The bulk of these subscribers are currently located in the western world, while the number of subscribers in Asia is forecasted to grow dramatically. However, the revenue from Asian markets is not as significant due to the lower average revenue per user, meaning that the most profitable customers will continue to reside in the United States and the European Economic Zone. Key partners include ISPs and IPTV providers. Cooperation among stakeholders would minimize the risk of legal disputes. In terms of demographic market segmentation, we believe that our target audience is similar to that of Slingbox. A rough estimate of their customers show that they are males, aged 20-50, residing in urban areas of the western world [QUANT]. As a consequence, we believe that in terms of behavioral segmentation, our customers are professionals and have high private mobility (residing in the course of a year in both second homes and hotels). Sizing the target audience in absolute terms lies outside the scope of this paper. Instead, we conclude that there exists an audience that is ready, willing, and able to pay for an IPv6 multicast proxy.

Selling directly to consumers using a strategy similar to the distribution of Slingbox could provide a substantial high-margin revenue stream. However, B2B-offers to ISPs would open a less volatile low-margin revenue-stream. By signing bulk-contracts with ISPs who can offer consumer oriented value propositions such as bundled deals a considerable sales volume could be generated. This strategy is currently used by the media company Voddler together with the Swedish ISP Bredbandsbolaget. [VODDLER].

6.2.4. Financing

The main initial revenue streams of a venture to offer a home proxy product would be revenues from sold goods and revenues from value added subscriptions. Costs include initial and continuous R&D, costs of goods sold, server costs, overhead costs, and customer acquisition costs.

It is possible that ISPs could save a great deal of money using multicast solutions. However, because of the high level of competition between ISPs, we believe that most service providers would pass the profit down the value chain by lowering bandwidth prices. More specifically, our proxy could actually raise the ISP’s costs since customers could stream vast amounts of data outside their ISP’s network using individual tunnels. This suggests that at some point the ISP would benefit by introducing RPs in networks that a lot of their subscribers are visiting.

7. Conclusions and Future work

7.1. Conclusions

The process of setting up a testing environment and testing the different protocols was a tedious and sometimes a frustrating process. However, some of the technologies we encountered during the project proved to be extremely mature and useful. The XORP-project has become a viable alternative to router hardware from Cisco and Juniper. It is robust, flexible, and relatively straightforward to get started with for administrators used to these other products. The Linux platform has matured within the context of virtualization thanks to the KVM-project. During our project we did not encounter any problems with KVM or our host machine running Ubuntu Linux. Our initial expectation was that the kernel space implementation would offer the best performance of the tested implementations. However, the kernel implementation was extremely tedious to get started with and we were disappointed with its poor performance. In other words, one should not immediately dismiss user-space applications as less capable than their low-level substitutes. As discussed in section 5.2, the virtual environment hard to benchmark due to its complexity. Therefore no tests with higher packet-loads were conducted. Furthermore, it is possible that performance may differ if the tests are conducted on dedicated hardware.

In accordance with our ambitions in section 1, our thesis demonstrates that IPv6 multicast proxying is a viable technology, and has been for quite some time. Several working solutions exist, with varying maturity and delay performance. The main issue with IPv6 multicast proxying is that it is currently a niche technology with limited real-life applications. Therefore community support is limited, and the development of most projects has stagnated during the past decade. However, several trends point to an increased interest in IPv6 multicast solutions. IPv6 migration has accelerated, especially in Asia. Furthermore, IPTV has been a commercial success. Therefore, we believe that new projects will be initiated to provide robust multicast forwarding capabilities in future IPv6 networks.

7.2. Future work

A suggestion for further research is to investigate why the kernel performs routing much slower than the user-level applications, and to suggest and implement modifications to improve the performance and stability of the code. As the tests were conducted in a virtualized environment they should be to re-run on dedicated hardware to investigate if any performance difference exists. If the performance difference is substantial, this should be investigated even further. Issues regarding ecmh usage with Linux kernels with IPv6 multicast forwarding enabled (as described in section 5.5) could also be studied.

It is feasible that XORP could be configured to work as an IPv6 multicast proxy. It would also be interesting to implement [RFC4605] using Linux kernel multicast forwarding. These two suggestions should be implemented together with thorough testing and benchmarking in the future.

Finally a potential future thesis would be studying multicast in Mobile IPv6 as specified in [RFC5757] or [MULTIMOB].

References

[6RD] W. Townsley and O. Troan, IPv6 Rapid Deployment on IPv4 Infrastructures (6rd), IETF Internet-Draft, May 2010 http://tools.ietf.org/html/draft-ietf-softwire-ipv6-6rd-10

[AICCU] http://www.sixxs.net/tools/aiccu/ Accessed on May 24th 2010

[AYIYA] http://www.sixxs.net/tools/ayiya/ Accessed on May 24th 2010

[BLAGA] T.Blaga, V.Dobrota, F.Szasz & R.Vidrascu, An on Demand IPv4/IPv6 Multicast Translator, 6th RoEduNet International Conference Networking in Education and Research, Craiova, Romania, November 23-24, 2007, pp.38-43 http://www.csfnau.kiev.ua/kipz/ua/ROMANIA/papers/038%20-%20043%20-%20Blaga.pdf

[CNGI] Ben Worthen, Internet Strategy: China's Next Generation Internet, CIO, CXO Media Inc., July 15, 2006 http://www.cio.com/article/22985/Internet_Strategy_China_s_Next_Generation_Internet_

[DRUMMONDS] Scott Drummonds, Inaccuracy of In-guest Performance Counters, February 2010 http://vpivot.com/2010/02/10/inaccuracy-of-in-guest-performance-counters/

[DURAND] A. Durand (Editor), Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion, IETF Internet Engineering Task Force, Internet-Draft, March 2010 http://tools.ietf.org/html/draft-ietf-softwire-dual-stack-lite-04

[GLANZMANN] R Glanzmann and J Fontanil, Multicast-Proxy-Server für Audio-Live-Streams, Hochschule für Technik Rapperswil, Rapperswil, Switzerland, July 2006 http://security.hsr.ch/projects/SA_2001_MulticastProxyServer-for-AudioLiveStreams.pdf

[HÅDEN] T. Hådén, IPv6 Home Automation, Bachelor's thesis, School of Information and Communication Technology, Royal Institute of Technology (KTH), Stockholm, Sweden, June 2009, TRITA-ICT-EX-2009:28

[HOLIDAY] J. Holliday, D. Agrawal, and A.E. Abbadi, Using Multicast Communication to Reduce Deadlocks in Replicated Databases, Proc. 19th Symp. Reliable Distributed Systems, pp. 196-205, 2000. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.3472&rep=rep1&type=pdf

[IPTV] IPTV Global Forecast (2008-2013), itve.org, International Television Expert Group, November 2009 http://www.international-television.org/tv_market_data/global-iptv-forecast-2009-2013.html

[JANNETEAU] C. Janneteau, E. Riou, A. Petrescu, A. Olivereau, and H.-Y. Lach, IPv6 Multicast for Mobile Networks with MLD-Proxy, Internet Draft, April 2004 http://tools.ietf.org/html/draft-janneteau-nemo-multicast-mldproxy-00

[JIANG1] Jiang Wu, A mobility support agent architecture for seamless IP handover, Licentiate thesis, Department of Teleinformatics (Institutionen för

teleinformatik), Royal Institute of Technology (KTH), KTH/IT/AVH--00/05--SE, Sept. 2000, 66 pages
http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-1083

[JIANG2] Jiang Wu and Gerald Q. Maguire Jr., Agent Based Seamless IP Multicast Receiver Handover, IFIP Conference on Personal Wireless Communications (PWC'2000), Gdansk, Poland, 14-15 September 2000.

[JIANG3] S. Jiang and D. Gu, Multicast Proxy in IPv6/IPv4 Transition, V6OPS Work Group, Internet Draft, March 1, 2010
http://tools.ietf.org/html/draft-jiang-behave-v4v6mc-proxy-00

[JUNIPER] MLD Proxy
http://www.juniper.net/techpubs/software/erx/junose53/swconfig-routing-vol1/html/ipv6-multicast-config14.html

[KVM] http://www.linux-kvm.org/
Accessed on May 24th 2010

[LINUX-IPV6] http://www.linux-ipv6.org/
Accessed on May 24th 2010

[MULTIMOB] IETF Multicast Mobility Working Group
https://datatracker.ietf.org/wg/multimob/charter/

[NARTEN] T. Narten, IETF Statement on IPv4 Exhaustion and IPv6 Deployment, IETF Network Working Group, Internet-Draft, November 2007
http://tools.ietf.org/html/draft-narten-ipv6-statement-00

[NILSSON] A. Nilsson and M. Lindberg, Virtually@Home, Bachelor's thesis, School of Information and Communication Technology, Royal Institute of Technology (KTH), Stockholm, Sweden, December 2009
TRITA-ICT-EX-2009:219

[QEMU] http://www.qemu.org/
Accessed on May 24th 2010

[QUANT] http://www.quantcast.com/slingbox.com
Accessed on May 24th 2010

[RFC1075] D. Waitzman, C. Partridge, and S. Deering, Distance Vector Multicast Routing Protocol, IETF Network Working Group, Request For Comments: 1075, November 1988
http://tools.ietf.org/html/rfc1075

[RFC1584] J. Moy, Multicast Extensions to OSPF, IETF Network Working Group, Request for Comments: 1584, March 1994
http://tools.ietf.org/html/rfc1584

[RFC2189] A. Ballardie, Core Based Trees (CBT version 2) Multicast Routing, IETF Network Working Group, Request for Comments: 2189, September 1997
http://tools.ietf.org/html/rfc2189

[RFC2201] A. Ballardie, Core Based Trees (CBT) Multicast Routing Architecture, IETF Network Working Group, Request for Comments: 2201, September 1997
http://tools.ietf.org/html/rfc22201

[RFC2460] S. Deering and R. Hinden, Internet Protocol, Version 6 (IPv6) Specification, IETF Network Working Group, Request for Comments: 2460, December 1998
http://tools.ietf.org/html/rfc2460

[RFC2464] M. Crawford, Transmission of IPv6 Packets over Ethernet Networks, IETF Network Working Group, Request for Comments: 2464, December 1998 http://tools.ietf.org/html/rfc2464

[RFC2710] S. Deering, W. Fenner, and B. Haberman, Multicast Listener Discovery (MLD) for IPv6, IETF Network Working Group, Request for Comments: 2710, October 1999 http://tools.ietf.org/html/rfc2710

[RFC2730] S. Hanna, B. Patel, and M. Shah, Multicast Address Dynamic Client Allocation Protocol (MADCAP), IETF Network Working Group, Requests for Comments: 2730, December 1999 http://tools.ietf.org/html/rfc2730

[RFC2784] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, Generic Routing Encapsulation (GRE), IETF Network Working Group, Requests for Comments: 2784, March 2000 http://tools.ietf.org/html/rfc2784

[RFC3056] B. Carpenter and K. Moore, Connection of IPv6 Domains via IPv4 Clouds, IETF Network Working Group, Request for Comments: 3056, February 2001 http://tools.ietf.org/html/rfc3056

[RFC3170] B. Quinn and K. Almeroth, IP Multicast Applications: Challenges and Solutions, IETF Network Working Group, Request for Comments: 3170, September 2001 http://tools.ietf.org/html/rfc3170

[RFC3306] B. Haberman and D. Thaler, Unicast-Prefix-based IPv6 Multicast Addresses, IETF Network Working Group, Request for Comments: 3306, August 2002 http://tools.ietf.org/html/rfc3306

[RFC3307] B. Haberman, Allocation Guidelines for IPv6 Multicast Addresses, IETF Network Working Group, Request for Comments: 3307, August 2002 http://tools.ietf.org/html/rfc3307

[RFC3376] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, Internet Group Management Protocol, Version 3, IETF Network Working Group, Request for Comments: 3376, October 2002 http://tools.ietf.org/html/rfc3376

[RFC3569] S. Bhattacharyya, Ed., An Overview of Source-Specific Multicast (SSM), IETF Network Working Group, Request for Comments: 3569, July 2003 http://tools.ietf.org/html/rfc3569

[RFC3956] P. Savola and B. Haberman, Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address, IETF Network Working Group,Request for Comments: 3956, November 2004 http://tools.ietf.org/html/rfc3956

[RFC3973] A. Adams, J. Nicholas, and W. Siadak, Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised), IETF Network Working Group, Request for Comments: 3973, January 2005 http://tools.ietf.org/html/rfc3973

[RFC4213] E. Nordmark and R. Gilligan, Basic Transition Mechanisms for IPv6 Hosts and Routers, IETF Network Working Group, Request for Comments: 4213, October 2005 http://tools.ietf.org/html/rfc4213

[RFC4291] R. Hinden and S. Deering, IP Version 6 Addressing Architecture, IETF Network Working Group, Request for Comments: 4291, February 2006 http://tools.ietf.org/html/rfc4291

[RFC4380] C. Huitema, Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs), IETF Network Working Group, Request for Comments: 4380, February 2006 http://tools.ietf.org/html/rfc4380

[RFC4389] D. Thaler, M. Talwar and C. Patel, Neighbor Discovery Proxies (ND Proxy), Network Working Group, Request for Comments: 4389, April 2006 http://tools.ietf.org/html/rfc4389

[RFC4443] A. Conta, S. Deering, and M. Gupta (Editor), Internet Control Message Protocol (ICMPv6) for the Internet Version 6 (IPv6) Specification, IETF Network Working Group, Request for Comments: 4443, March 2006 http://tools.ietf.org/html/rfc4443

[RFC4601] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised), IETF Network Working Group, Request for Comments: 4601, August 2006 http://tools.ietf.org/html/rfc4601

[RFC4605] B. Fenner, H. He, B. Haberman, and H. Sandick, Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP/MLD Proxying"), IETF Network Working Group, Request for Comments: 4605, August 2006 http://tools.ietf.org/html/rfc4605

[RFC4861] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, Neighbor Discovery for IP version 6 (IPv6) , IETF Network Working Group, Request for Comments: 4861, September 2007 http://tools.ietf.org/html/rfc4861

[RFC4862] S. Thomson, T. Narten, and T. Jinmei, IPv6 Stateless Address Autoconfiguration, IETF Network Working Group, Request for Comments: 4862, September 2007 http://tools.ietf.org/html/rfc4862

[RFC5569] R. Despres, IPv6 Rapid Deployment on IPv4 Infrastructures (6rd), IETF Independent Submission, Request for Comments: 5569, January 2010 http://tools.ietf.org/html/rfc5569

[RFC5757] T. Schmidt, M. Waehlisch, and G. Fairhurst, Multicast Mobility in Mobile IP Version 6 (MIPv6): Problem Statement and Brief Survey, Internet Research Task Force, Request for Comments: 5757, February 2010

[RFC5771] M. Cotton, L. Vegoda and D. Meyer, IANA Guidelines for IPv4 Multicast Address Assignment, IETF Best Current Practice 51, March 2010 http://tools.ietf.org/html/rfc5771

[SLING] http://www.slingbox.com/ Accessed on May 24th 2010

[TIC] http://www.sixxs.net/tools/tic/ Accessed on May 24th 2010

[VENAAS] S. Venaas, H. Asaeda, S. Suzuki, and T. Fujisaki, An IPv4 - IPv6 multicast translator, IETF Network Working Group, Internet-Draft, July 2009 http://tools.ietf.org/html/draft-venaas-behave-mcast46-01

[VODDLER] Bredbandsbolaget provfilmar för Voddler, Press release, November 12, 2009 http://www.bredbandsbolaget.se/published_images/20091112_prm%20Voddler%20kampanj%20final.pdf

[XORP] http://www.xorp.org/

[ZMAAP] O. Catrina (Editor), D. Thaler, and E. Guttman, IETF Network Working Group, Internet Draft, Zeroconf Multicast Address Allocation Protocol (ZMAAP), October 2002 http://tools.ietf.org/html/draft-ietf-zeroconf-zmaap-02

Appendix A – mcast.py

```
#!/usr/bin/env python
#
# Send/receive UDP multicast packets.
# Requires that your OS kernel supports IP multicast.
#
# Based on
http://svn.python.org/projects/python/trunk/Demo/sockets/mcast.py
#
# PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2
#
# Usage:
#   mcast -s (sender, IPv4)
#   mcast -s -6 (sender, IPv6)
#   mcast    (receivers, IPv4)
#   mcast  -6  (receivers, IPv6)

MYPORT = 8123
MYGROUP_4 = '225.0.0.250' # Random addresses used for testing
MYGROUP_6 = 'ff15:7079:7468:6f6e:6465:6d6f:6d63:6173'
MYTTL = 10 # Increase to reach other networks

import time
import struct
import socket
import sys

def main():
    group = MYGROUP_6 if "-6" in sys.argv[1:] else MYGROUP_4

    if "-s" in sys.argv[1:]:
        sender(group)
    else:
        receiver(group)


def sender(group):
    addrinfo = socket.getaddrinfo(group, None)[0]

    s = socket.socket(addrinfo[0], socket.SOCK_DGRAM)

    # Set Time-to-live (optional)
    ttl_bin = struct.pack('@i', MYTTL)
    if addrinfo[0] == socket.AF_INET: # IPv4
        s.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL,
ttl_bin)
    else:
        s.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_MULTICAST_HOPS,
ttl_bin)

    while True:
        data = repr(time.time())
        s.sendto(data + '\0', (addrinfo[4][0], MYPORT))
        time.sleep(1)
```

```
def receiver(group):
    # Look up multicast group address in name server and find out IP
version
    addrinfo = socket.getaddrinfo(group, None)[0]

    # Create a socket
    s = socket.socket(addrinfo[0], socket.SOCK_DGRAM)

    # Allow multiple copies of this program on one machine
    # (not strictly needed)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    # Bind it to the port
    s.bind(('', MYPORT))

    group_bin = socket.inet_pton(addrinfo[0], addrinfo[4][0])
    # Join group
    if addrinfo[0] == socket.AF_INET: # IPv4
        mreq = group_bin + struct.pack('=I', socket.INADDR_ANY)
        s.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP,
mreq)
    else:
        mreq = group_bin + struct.pack('@I', 0)
        # mreq = group_bin + struct.pack('@I', 5)  # On proxynet2
        s.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_JOIN_GROUP,
mreq)

    # Loop, printing any data we receive
    while True:
        data, sender = s.recvfrom(1500)
        while data[-1:] == '\0': data = data[:-1] # Strip trailing
\0's
        print (str(sender) + '  ' + repr(data))


if __name__ == '__main__':
    main()
```

Appendix B – forwarder.py

```
#!/usr/bin/env python
#
# Forwards UDP multicast packets.
# Requires that your OS kernel supports IP multicast.
#
# Usage:
#   forwarder     (IPv4)
#   forwarder -6  (IPv6)

MYPORT = 8123
MYGROUP_4 = '225.0.0.250'
MYGROUP_6 = 'ff15:7079:7468:6f6e:6465:6d6f:6d63:6173'
MYTTL = 1 # Increase to reach other networks

import time
import struct
import socket
import sys

def main():
    group = MYGROUP_6 if "-6" in sys.argv[1:] else MYGROUP_4

    # Look up multicast group address in name server and find out IP
version
    addrinfo = socket.getaddrinfo(group, None)[0]

    # Create two sockets
    sin = socket.socket(addrinfo[0], socket.SOCK_DGRAM)
    sout = socket.socket(addrinfo[0], socket.SOCK_DGRAM)

    # Allow multiple connections to the port
    sin.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sout.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    # Bind it to the port
    sin.bind(('', MYPORT))
    sout.bind(('', MYPORT))

    # Set Time-to-live
    ttl_bin = struct.pack('@i', MYTTL)
    if addrinfo[0] == socket.AF_INET: # IPv4
        sout.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_IF,
socket.inet_aton('10.0.3.1'))
        sout.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL,
ttl_bin)
    else:
        sout.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_MULTICAST_IF,
struct.pack("I", 4))
        sout.setsockopt(socket.IPPROTO_IPV6,
socket.IPV6_MULTICAST_HOPS, ttl_bin)
```

```
    group_bin = socket.inet_pton(addrinfo[0], addrinfo[4][0])

    # Join group
    if addrinfo[0] == socket.AF_INET: # IPv4
        mreq = group_bin + struct.pack('=I', socket.INADDR_ANY)
        sin.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP,
mreq)
    else:
        mreq = group_bin + struct.pack('@I', 2)
        sin.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_JOIN_GROUP,
mreq)

    # Loop, forwarding any data we receive
    while True:
        data, sender = sin.recvfrom(1500)
        sout.sendto(data + '\0', (addrinfo[4][0], MYPORT))

if __name__ == '__main__':
    main()
```

Appendix C – Startup script

```
#!/bin/sh
#most of the code utilized (with permission) from Elis Kullberg, 2009

USERID=`whoami`

#initialize TAP-devices

iface0=`tunctl -b -u $USERID`
iface1=`tunctl -b -u $USERID`
iface2=`tunctl -b -u $USERID`
iface3=`tunctl -b -u $USERID`
iface4=`tunctl -b -u $USERID`
iface5=`tunctl -b -u $USERID`
iface6=`tunctl -b -u $USERID`

echo "Crated 5 interfaces"

echo "tun0 is " $iface0
echo "tun1 is " $iface1
echo "tun2 is " $iface2
echo "tun3 is " $iface3
echo "tun4 is " $iface4
echo "tun5 is " $iface5
echo "tun6 is " $iface6

# Add first tap-device to existing bridge (0) (connected to the
world)
ifconfig $iface0 0.0.0.0 up
brctl addif br0 $iface0
echo $iface0 "to br0"

# Add new TAP-devices to new bridges
ifconfig $iface1 0.0.0.0 up
ifconfig $iface2 0.0.0.0 up
brctl addbr br1
ifconfig br1 0.0.0.1 up
brctl addbr br2
ifconfig br2 0.0.0.1 up
brctl addif br1 $iface1
echo $iface1 to br1
brctl addif br2 $iface2
echo $iface2 to br2
ifconfig $iface3 0.0.0.0 up
ifconfig $iface4 0.0.0.0 up
brctl addif br1 $iface3
echo $iface3 to br1
brctl addif br2 $iface4
echo $iface4 to br2
brctl addbr br3
ifconfig br3 0.0.0.1 up
ifconfig $iface5 0.0.0.0 up
ifconfig $iface6 0.0.0.0 up
brctl addif br3 $iface5
brctl addif br3 $iface6
```

```
# Generate some random MAC addresses
# A special thanks to pheldens @ qemu forums for this
ranmac0=$(echo -n DE:AD:BE:EF ; for i in `seq 1 2` ; \
do echo -n `echo ":$RANDOM$RANDOM" | cut -n -c -3` ;done)
ranmac1=$(echo -n DE:AD:BE:EF ; for i in `seq 1 2` ; \
do echo -n `echo ":$RANDOM$RANDOM" | cut -n -c -3` ;done)
ranmac2=$(echo -n DE:AD:BE:EF ; for i in `seq 1 2` ; \
do echo -n `echo ":$RANDOM$RANDOM" | cut -n -c -3` ;done)
ranmac3=$(echo -n DE:AD:BE:EF ; for i in `seq 1 2` ; \
do echo -n `echo ":$RANDOM$RANDOM" | cut -n -c -3` ;done)
ranmac4=$(echo -n DE:AD:BE:EF ; for i in `seq 1 2` ; \
do echo -n `echo ":$RANDOM$RANDOM" | cut -n -c -3` ;done)
ranmac5=$(echo -n DE:AD:BE:EF ; for i in `seq 1 2` ; \
do echo -n `echo ":$RANDOM$RANDOM" | cut -n -c -3` ;done)

serveroptions="-localtime -m 512 -cdrom XORP-1.6-LiveCD.iso"
echo "randmacs + serverops"

echo "starting KVM"
kvm -usb -usbdevice host:090c:1000 -net nic,vlan=0,macaddr=$ranmac0 -
net tap,vlan=0,ifname=$iface0 -net
nic,vlan=1,macaddr=de:ad:be:ef:18:28 -net tap,vlan=1,ifname=$iface1 -
net nic,vlan=2,macaddr=de:ad:be:ef:28:86 -net
tap,vlan=2,ifname=$iface2 -net nic,vlan=3,macaddr=de:ad:be:ef:38:81 -
net tap,vlan=3,ifname=$iface6 $serveroptions&

clientoptions="-localtime -m 256"

kvm -usb -usbdevice host:1307:0163 -hda proxy1.img -net
nic,vlan=3,macaddr=$ranmac3 -net tap,vlan=3,ifname=$iface3
$clientoptions&
kvm -hda proxy2.img -net nic,vlan=4,macaddr=$ranmac4 -net
tap,vlan=4,ifname=$iface4 $clientoptions&
kvm -hda proxy3.img -net nic,vlan=5,macaddr=$ranmac5 -net
tap,vlan=5,ifname=$iface5 $clientoptions&

echo "Press enter to tear down"
read stuff

#tear down the virtual devices
tunctl -d tap0 &> /dev/null
tunctl -d tap1 &> /dev/null
tunctl -d tap2 &> /dev/null
tunctl -d tap3 &> /dev/null
tunctl -d tap4 &> /dev/null
tunctl -d tap5 &> /dev/null
tunctl -d tap6 &> /dev/null
echo "TAP 0-5 gone"
# Remove redundant virtual bridge
brctl delbr br1
brctl delbr br2
brctl delbr br3
echo "Attempted to remove br 1-3"
```

Appendix D – XORP configuration

```
/*XORP Configuration File, v1.0*/
protocols {
    fib2mrib {
        disable: false
    }
    igmp {
        disable: false
        interface re1 {
            vif re1 {
                disable: false
                version: 2
                enable-ip-router-alert-option-check: false
                query-interval: 125
                query-last-member-interval: 1
                query-response-interval: 10
                robust-count: 2
            }
        }
        interface re2 {
            vif re2 {
                disable: false
                version: 2
                enable-ip-router-alert-option-check: false
                query-interval: 125
                query-last-member-interval: 1
                query-response-interval: 10
                robust-count: 2
            }
        }
        traceoptions {
            flag {
                all {
                    disable: false
                }
            }
        }
    }
    mld {
        disable: false
        interface re1 {
            vif re1 {
                disable: false
                version: 1
                enable-ip-router-alert-option-check: false
                query-interval: 125
                query-last-member-interval: 1
                query-response-interval: 10
                robust-count: 2
            }
        }
```

```
    interface re2 {
        vif re2 {
            disable: false
            version: 1
            enable-ip-router-alert-option-check: false
            query-interval: 125
            query-last-member-interval: 1
            query-response-interval: 10
            robust-count: 2
        }
    }
    traceoptions {
        flag {
            all {
                disable: false
            }
        }
    }
}
pimsm4 {
    disable: false
    interface re1 {
        vif re1 {
            disable: false
            dr-priority: 1
            hello-period: 30
            hello-triggered-delay: 5
        }
    }
    interface re2 {
        vif re2 {
            disable: false
            dr-priority: 1
            hello-period: 30
            hello-triggered-delay: 5
        }
    }
    interface "register_vif" {
        vif "register_vif" {
            disable: false
            dr-priority: 1
            hello-period: 30
            hello-triggered-delay: 5
        }
    }
    bootstrap {
        disable: false
        cand-rp {
            group-prefix 224.0.0.0/4 {
                is-scope-zone: false
                cand-rp-by-vif-name: "re1"
                cand-rp-by-vif-addr: 0.0.0.0
                rp-priority: 192
                rp-holdtime: 150
            }
        }
    }
}
```

```
pimsm6 {
    disable: false
    interface re1 {
        vif re1 {
            disable: false
            dr-priority: 1
            hello-period: 30
            hello-triggered-delay: 5
        }
    }
    interface re2 {
        vif re2 {
            disable: false
            dr-priority: 1
            hello-period: 30
            hello-triggered-delay: 5
        }
    }
    interface "register_vif" {
        vif "register_vif" {
            disable: false
            dr-priority: 1
            hello-period: 30
            hello-triggered-delay: 5
        }
    }
    bootstrap {
        disable: false
        cand-rp {
            group-prefix ff00::/8 {
                is-scope-zone: false
                cand-rp-by-vif-name: "re1"
                cand-rp-by-vif-addr: ::
                rp-priority: 192
                rp-holdtime: 150
            }
        }
    }
}
static {
    disable: false
    route 10.0.0.0/24 {
        next-hop: 10.0.0.2
        metric: 1
    }
    route 10.0.1.0/24 {
        next-hop: 10.0.1.2
        metric: 1
    }
    route 10.0.2.0/24 {
        next-hop: 10.0.2.2
        metric: 1
    }
    route fc00::/64 {
        next-hop: fc00::2
        metric: 1
    }
    route fc00:1::0/64 {
        next-hop: fc00:1::2
        metric: 1
    }
```

```
    }
}
fea {
    unicast-forwarding4 {
        disable: false
    }
    unicast-forwarding6 {
        disable: false
    }
}
interfaces {
    restore-original-config-on-shutdown: false
    interface discard0 {
        description: "discard interface"
        disable: false
        discard: true
        unreachable: false
        management: false
        vif discard0 {
            disable: false
            address 192.0.2.1 {
                prefix-length: 32
                disable: false
            }
        }
    }
    interface re0 {
        description: "mot natet"
        disable: false
        discard: false
        unreachable: false
        management: false
        vif re0 {
            disable: false
        }
    }
    interface re1 {
        description: "subnat 1"
        disable: false
        discard: false
        unreachable: false
        management: false
        vif re1 {
            disable: false
            address 10.0.0.1 {
                prefix-length: 24
                broadcast: 10.0.0.255
                disable: false
            }
            address fc00::1 {
                prefix-length: 64
                disable: false
            }
            address fe80::dead:beef:1828 {
                prefix-length: 64
                disable: false
            }
        }
    }
```

```
    interface re2 {
        description: "subnat 2"
        disable: false
        discard: false
        unreachable: false
        management: false
        vif re2 {
            disable: false
            address 10.0.1.1 {
                prefix-length: 24
                broadcast: 10.0.1.255
                disable: false
            }
            address fc00:1::1 {
                prefix-length: 64
                disable: false
            }
            address fe80::dead:beef:2886 {
                prefix-length: 64
                disable: false
            }
        }
    }
    interface lo0 {
        description: "Loopback interface"
        disable: false
        discard: false
        unreachable: false
        management: false
        vif lo0 {
            disable: false
        }
    }
    interface re3 {
        description: "subnat 3"
        disable: false
        discard: false
        unreachable: false
        management: false
        vif re3 {
            disable: false
            address 10.0.2.1 {
                prefix-length: 24
                broadcast: 10.0.2.255
                disable: false
            }
        }
    }
}
plumbing {
    mfea4 {
        disable: false
        interface re1 {
            vif re1 {
                disable: false
            }
        }
        interface re2 {
            vif re2 {
                disable: false
            }
```

```
        }
        interface "register_vif" {
            vif "register_vif" {
                disable: false
            }
        }
        traceoptions {
            flag {
                all {
                    disable: true
                }
            }
        }
    }
    mfea6 {
        disable: false
        interface re1 {
            vif re1 {
                disable: false
            }
        }
        interface re2 {
            vif re2 {
                disable: false
            }
        }
        interface "register_vif" {
            vif "register_vif" {
                disable: false
            }
        }
        traceoptions {
            flag {
                all {
                    disable: false
                }
            }
        }
    }
}
```

Appendix E – Network interface configuration for proxynet2

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
address 10.0.2.2
gateway 10.0.2.1
netmask 255.255.255.0
network 10.0.2.0
broadcast 10.0.2.255

# The old 6in4-tunnel
#auto 6in4
#iface 6in4 inet6 v4tunnel
#address fc00:2::2
#netmask 64
#endpoint 10.0.1.2
#post-up ip tunnel change 6in4 ttl 64

auto tunnel0
iface tunnel0 inet6 static
address fc00:3::2
netmask 64
pre-up ip tunnel add tunnel0 mode gre remote 10.0.1.2 local 10.0.2.2
ttl 64
post-down ip tunnel del tunnel0

iface tunnel0 inet static
address 10.0.3.2
netmask 255.255.255.0
network 10.0.3.0
broadcast 10.0.3.255
```

Appendix F – Computer Hardware information

Hostname: Magnus

Dell m1330-laptop

Intel Core 2 Duo 2.0 GHz

2 GB of RAM-memory

Ubuntu Linux 10.04 Beta with Linux Kernel 2.6.32

Appendix G – SMCroute bug report

Reporter: Elis Kullberg, elisk@kth.se, Hannes Junnila, haju@kth.se

Product: smcroute

Version: 0.94.1

Operating system: Ubuntu 9.10 server, kernel 2.6.31-20 compiled with mc_forwarding enabled.

Severity:
Major

Summary:
Tunnel devices are not initiated as multicast capable, even though their multicast-flag is set using ifconfig before initializing smcroute.

Description:

Reproduce steps:
1. Initialize tunnel (for example IPv6 in IPv4) using ip
2. Set multicast flag using ifconfig.or ip
3. Start smcroute daemon and specify route + devices

Expected result: Interfaces active in /proc/net/ip6_mr_vif and a working static IPv6 multicast route.

Actual result: Only eth0 active in /proc/net/ip6_mr_vif. Other interfaces have /proc/sys/net/ipv6/conf/[interface]/mc_forwarding set to “0”.

Appendix H – ecmh.dump

```
*** Subscription Information Dump

Group : ff15:7079:7468:6f6e:6465:6d6f:6d63:6173
      Bytes  : 7316
      Packets: 110
      Interface: tunnel0 (1)
            :: INCLUDE (99 seconds old)

*** Subscription Information Dump (end - 1 groups, 1 subscriptions)

*** Interface Dump

Interface: eth0
  Index number           : 2
  MTU                    : 1500
  Interface Type         : Ethernet (1)
  Link-local address     : fe80::dcad:beff:feef:0
  Global unicast address : fc00:1::2
  MLD version            : v1
  Packets received       : 145
  Packets sent           : 1
  Bytes received         : 9842
  Bytes sent             : 72
  ICMP's received        : 8
  ICMP's sent            : 1

Interface: tunnel0
  Index number           : 5
  MTU                    : 1476
  Interface Type         : Unknown (778)
  Link-local address     : fe80::56789:1234:0
  Global unicast address : fc00:3::1
  MLD version            : v2
  Packets received       : 9
  Packets sent           : 106
  Bytes received         : 624
  Bytes sent             : 7070
  ICMP's received        : 6
  ICMP's sent            : 2

*** Interface Dump (end - 2 interfaces)

*** Statistics Dump

Version              : ecmh 2005.02.09
Started              : 2010-05-12 11:44:38 GMT
Uptime               : 0 days 00:02:27

Interfaces Monitored : 2
Groups Managed       : 1
Total Subscriptions  : 1
v2 Robustness Factor : 2
Subscription Timeout : 250
```

```
Packets Received     : 154
Packets Sent         : 107
Bytes Received       : 10466
Bytes Sent           : 7142
ICMP's received      : 14
ICMP's sent          : 3
Hop Limit Exceeded   : 5

*** Statistics Dump (end)
```

TRITA-ICT-EX-2010:106

www.kth.se