Project Number:	**IST-2001-32603**
Project Title:	**6NET**
CEC Deliverable Number:	**32603/UOS/DS/5.9/A1**
Contractual Date of Delivery to the CEC:	30th September 2004
Actual Date of Delivery to the CEC:	4th January 2005
Title of Deliverable:	D5.9: Report on testing application over PIM-SSM deployment
Work package contributing to Deliverable:	WP5
Version:	1.0 (4[h] January 2005)
Type of Deliverable*:	R
Deliverable Security Class**:	PU
Reviewer:	Brian Carpenter (IBM)
Editor:	Tim Chown (University of Southampton)
Contributors:	Tim Chown (University of Southampton), Nick Lamb (University of Southampton), Stig Venaas (UNINETT), Mickaël Hoerdt (University of Strasbourg), Savvas Th. Anastasiades (GRNET), Piers O'Hanlon (UCL), Maciej Strozyk (PSNC), Simon Leinen (SWITCH).

* Type: P - Prototype, R - Report, D - Demonstrator, O - Other

** Security Class: PU- Public, PP – Restricted to other programme participants (including the Commission), RE – Restricted to a group defined by the consortium (including the Commission), CO – Confidential, only for members of the consortium (including the Commission)

Abstract:

In this document we describe the theory, implementation, configuration and deployment issues for Source-Specific Multicast (SSM) with IPv6 and we report on testing experience.

Keywords:

IPv6, Multicast, Source Specific Multicast, SSM

Executive Summary

This document describes the current status of IPv6 and Source Specific Multicast (SSM), based on the experiences of running trial networks and applications in the 6NET project. SSM is an alternative multicast architecture to traditional Any Source Multicast (ASM); it is simpler and thus generally easier to deploy and debug, however there are some limitations in what SSM applications can reasonably do.

Support for SSM in vendor and open source platforms is growing in stature. In our trials we were able to connect eight sites with a variety of such platforms, verifying the basic operation of the protocol, and demonstrating a new reliable file transfer protocol (FLUTE) with an application called Mad, which the project ported so support IPv6 SSM.

The project has also defined other tools, in particular an 'SSMifier' that can act as an SSM-ASM 'converter' for applications. This is described in the document.

Further work is planned before the end of the 6NET project, and this will be described in an updated deliverable, due by June 2005.

Table of Contents

Table of Figures

1. Introduction

Source-Specific Multicast is a more recent form of IP Multicast compared to the "traditional" Any Source Multicast (ASM) model. In making a fundamental assumption that there is only a single source in a specific Multicast group, simplifications can be made that lead to a new form of multicast that should be easier to deploy and troubleshoot than ASM.

The 6NET project has been at the forefront of SSM testing and deployment. In this text we describe the theory, practice, implementation, testing and issues involved. The document is thus a "cookbook" for SSM with IPv6.

2. Source Specific Multicast (SSM)

In this section we discuss specifics of SSM.

2.1. Specification

Source-Specific Multicast (SSM) is defined within the Internet Engineering Task Force (IETF). A list of SSM-related IETF standards and drafts is maintained by SWITCH [SSM-Refs].

Some of the key IETF documents are:

- The SSM Architecture [SSM-Arch]

- Multicast deployment issues (general) [DEPLOY]

- An Introduction to SSM [SSM-Intro]

- The SSM API, defined within RFC3678 [SSM-API]

- IPv6 SSM address range (under ff3x::/32) [SSM-ADDR]

These documents together provide an excellent coverage of standardisation of IPv6 SSM (and the issues surrounding the protocol), though some are not SSM or IPv6 specific.

2.1.1. SSM basic principle

The traditional multicast service model allows for applications to join a group (referred to as "G" in most multicast texts) in order to receive any data sent to the group. A source simply sends IP datagrams to the multicast IP address of G. An application does not need to know which sources are available. This model is called any-source multicast (ASM).

In contrast, the Source-Specific Multicast (SSM) service model is based on applications joining so-called channels rather than groups. A channel is a pair (S,G) consisting of a unicast source address S and a multicast group address G. For each source S and group G the application is to receive data from, it must join the channel (S,G).

This means that the application explicitly specifies the sources, and hence it must know the source addresses.

2.1.2.   SSM technical specification details

In order to offer the SSM service, a number of components must be available. There must be a way for an application to tell the operating system that it wants to join a channel. The host (operating system) must have a way of signalling to the network, and finally the network must be able to do the necessary multicast routing and forwarding. Before looking at each of these components, we will also talk about multicast addresses.

2.1.2.1. Multicast addresses

For both IPv4 and IPv6 there are address ranges reserved for SSM use. For IPv6 the range is FF3x::/32 [SSM-ADDR] and for IPv4 it is 232/8 (i.e. all addresses starting with 232). Note that one can also join specific sources and make use of the service model also for other addresses. In that case some receivers may join a specific source, while others may join the group. In general one should use an SSM address if the session is intended for SSM usage only.

2.1.2.2. SSM routing

We will here look at what goes on between the multicast routers. The forwarding of multicast packets in itself is no different for SSM than for ASM. The routing is different though. The most popular multicast routing protocol is PIM-SM [PIM-SM]. When a host joins a multicast group, a multicast tree (a so called shared tree) is built from a router in the network configured to be a Rendezvous Point (RP) towards the receiver.

When a source starts sending to the group, the packets are sent from a router on the source's local network. Initially the multicast packets are actually encapsulated into unicast packets and sent as so called PIM register messages from the router next to the source to the RP.

In this way the packets are forwarded from the source to the receivers via the RP. However, once the multicast packets reach the last-hop router (the last router before a receiving host), the last-hop router can shortcut this by building so-called shortest-path trees towards the sources. The last-hop router learns the source addresses by looking at the source addresses of the multicast packets.

There are several problems with the above. One issue is how to make sure all routers agree on where the RP is for a group. Protocols like BSR can be used for maintaining these RP-to-group mappings in the routers. This is very hard to do throughout the Internet though for several reasons. There are two solutions to this. For IPv4 there is [MSDP] which actually allows for different parts of the internet to use different RPs for the same group. This is done by exchanging source information between RPs.

MSDP is not available for IPv6, but there is a solution called [embedded-RP] that embeds RP-addresses into multicast addresses. There are also routers that have problems doing the PIM register encapsulation and decapsulation efficiently. Finally this architecture is quite complex, which in turn makes it hard to manage.

PIM-SM was originally created to provide the ASM service. It can also provide for the SSM service though. When a host joins a channel (S,G) the last-hop router immediately knows the source address S, so without waiting for multicast packets, it can immediately start building the shortest-path tree towards the source. This process is exactly the same as described above. Hence PIM-SM can be used without modifications to provide SSM. In order to provide SSM service only, no RPs are necessary. This basically removes all the complexities. There is no need for BSR, MSDP etc. There are also no shared trees, no PIM register with encapsulation and decapsulation. This means that an SSM-only network is much easier to manage.

The only specific support a PIM router should have for SSM, is that for the SSM-specific group ranges, it should only allow SSM usage. This involves ignoring so called (*,G)-joins, that is joins that are not specific and used for building shared trees, and also not sending and ignoring PIM register messages when G is inside an SSM range.

Note that when using PIM-SM, SSM and ASM can co-exist in the same network. This means that one can easily deploy and offer both together.

2.1.2.3. Communication between routers and hosts

For IPv4 and IPv6, there are protocols IGMP and MLD respectively that are used between routers and hosts for signalling which groups the host wants to receive data for. The latest versions of these protocols, [IGMPv3] and [MLDv2] also let the host specify specific sources. This is exactly what is needed for SSM. In addition to SSM, they also allow a host to join a group as in ASM, but blocking certain sources.

The router should only forward multicast packets onto the edge network if there are hosts interested in the packet's source and group. This means that if G is an SSM group, it should only forward it if at least one host has joined the channel (S,G) where S is the source address. If G is not an SSM group, it may also forward it if there is at least one host that has joined the group G (without specifying the source) and not blocked the source.

Despite the above, a host may receive traffic from a source it has blocked or not specifically joined. One way this can happen, is where different hosts join different sources for the same group. Thus the host must also be able to do filtering. In the extreme case there may be one application joining different sources for the same group on two different sockets. The host will then signal interest for both sources to the router, and must itself take care to return the packets on the appropriate sockets. We will discuss the host internal workings further in the next section.

One of the main obstacles to SSM deployment has been the availability of these protocols. This is now improving with several implementations. At least for IPv4 a few workarounds are available for some vendors. One example is telling the router that every time someone joins a group G it is to be treated as a join for (S,G).

Another obstacle may be switches. There are some switches that do IGMPv2 snooping that does not work well with IGMPv3. For IPv6 this may be less of a problem since very few support MLD snooping at all, although not having MLD snooping support may result in problems for multicast in general. There are some other ways for switches to limit flooding of multicast, but snooping seems to be the most widely used.

Note that for sending multicast the above protocols are not needed. A host does not in any way need to join and receive multicast in order to send.

2.1.2.4. SSM and applications

Network applications typically use the so-called "socket API" for communicating with the operating system. Some applications are programmed using a higher level interface, but the interaction with the operating system is still usually done using sockets. There are standard ways for an application to create a socket, and tell the operating system that it wants to join a specific group on that socket.

To make use of SSM, a new API is specified in [MSFAPI] that allows the application to also specify sources. Joining specific source is called "inclusion mode". It also allows "exclusion mode" which means join a group G (like in ASM) but exclude/block certain sources. This is similar to the features provided in IGMPv3 and MLDv2. The new API also makes it easier to write

protocol independent code (that is, code that works with both IPv4 and IPv6), so even when only joining a group G without specifying sources, the new API may be useful.

Porting applications from ASM to SSM can be simple. At least when the source addresses are available ahead of time. Supporting dynamic sources is much more complicated. There are also some specific considerations for applications using RTP/RTCP, see [RTCPSSM].

2.2. Comparison with Any Source Multicast (ASM)

We have already discussed some of these issues in Section 2.1.2.2.

SSM gives some protection against rogue sources sending to a multicast group. A content provider can announce its channels (S1,G), ..., (Sn,G) and an application joining those channels, will not receive packets sent to G with other source addresses than S1, ..., Sn. Also note that due to the so-called RPF check used in multicast distribution, there is protection against source address spoofing.

Since one joins channels rather than groups, it is not a problem if two different sources use the same group for two different sessions. This follows from the above. Someone joining one of the channels will not receive from the other, even if the group is the same. This means that one only needs to ensure uniqueness of the group used within the host. With ASM one needs different applications on different hosts to use different groups for different sessions. This can be hard to ensure. The fact that the receiver joins explicit sources allows for great simplifications of the multicast routing protocols, e.g. for PIM-SM one does not need a Rendezvous Point. We'll go into more technical details later.

All the advantages of SSM follows from the fact that the application explicitly specifies the source addresses. This is also the main problem with SSM. There are several applications where it is difficult to know in advance what sources should be joined. As will be described later, there are some ways to cope with this, basically doing the same job as PIM-SM's Rendezvous Point at the application level. The one use of multicast that is basically impossible to satisfy with SSM is service discovery, e.g. for locating IPv6 DHCP servers in a site, there is a standardized address FF05::1:3. This address can be hard-coded in a DHCP relay agent, and without any knowledge of the unicast addressses used in the site, it can simply send to this group.

2.3. Future IPv6 SSM Networks

Current opinion in the IETF multicast deployment working group (mboned) is largely agreed on SSM as the way forward. An open question is to what extent ASM may be needed as a supplement, e.g. for multi-party multicast applications.

SSM support in core routers is good; the current issue is MLDv2 support at the edges.

3. Configuring Available Implementations

3.1. Router platforms

3.1.1. Cisco

Support in IOS is good, with most platforms including SSM capability.   There are some exceptions, like the Catalyst 4500, 6500 and Cisco 10720 series.

3.1.1.1. Status

For example, IPv6 SSM is supported in Cisco 7200 routers as of at least 12.3(4)T. Full details can be found on the Cisco CCO Feature Navigator for those with CCO accounts.

3.1.1.2. Configuration example

Support can typically be turned on by a command as simple as:

```
ipv6 multicast-routing
```

Further configuration may be required for MBGP or scoped borders; these are discussed in the WP3 deliverable on multicast inter-domain routing.

3.1.2. BSD

3.1.2.1. Status

Versions of FreeBSD patched with the latest IPv6 tree from the KAME project include a working PIM SSM and MLDv2 router implementation with some limitations. In particular it does not cope with interfaces that are created or destroyed during the lifetime of the daemon.

Unpatched versions of FreeBSD include an IPv6 capable PIM implementation, but the version provided is not compiled to support SSM. In some versions of FreeBSD the OS does not provide definitions for the MLDv2 protocol, but a suitable definition can be made externally and if the PIM6SD is recompiled with this definition support for SSM is enabled.

Some patches from the KAME tree are needed to fix errors in the PIM SSM implementation. A patched version has been made available to 6NET participants for testing purposes.

3.1.2.2. Configuration example

An example is:

```
/etc/pim6sd.conf

phyint fxp0 mld_version any;

phyint dc0 mld_version any;

phyint vlan0 mld_version any;

phyint vlan1 mld_version any;

phyint vlan2 mld_version any;
```

3.1.3. Linux

3.1.3.1. Status

Experimental IPv6 multicast routing support is available for Linux. Kernel patches are available from:

http://clarinet.u-strasbg.fr/~hoerdt/linux_ipv6_mforwarding/

The pim6sd daemon is ported from BSD and is available at:

> http://clarinet.u-strasbg.fr/~hoerdt/pim6sd_linux/

3.1.3.2. Configuration example

After compiling kernel and daemon, one can run the daemon with configuration similar to what is done on BSD. So as described in 3.1.2.2 one should set the MLD version to "any" on the different physical interfaces.

3.1.4. 6WIND

3.1.4.1. Status

IPv6 SSM is supported as of at least 6WINDGate SixOS 6121 version 6.6.1 (s0).

3.1.4.2. Configuration example

The 6WIND device has not been tested in the 6NET project as yet.

3.1.5. Juniper

3.1.5.1. Status

Support is present in at least JUNOS 6.2.

3.1.5.2. Configuration example

No 6NET edge sites are using Juniper routers for SSM at this time.

3.1.6. Hitachi

3.1.6.1. Status

Hitachi OS (S-9181-61X 07-05-/A) supports PIM-SSM, this was supported from at least Ver.07-03.

3.1.6.2. Configuration example

No 6NET edge sites are using Hitachi routers for SSM at this time

3.2. Host platforms

3.2.1. Linux

3.2.1.1. Status

The Linux kernel itself has included the MLDv2 APIs needed for IPv6 SSM since 2.4.22 but some misunderstanding prevented the IPv6 stack from using the correct MLDv2 packet type in versions prior to 2.4.27-pre2 or 2.6.6-rc2. As a result of tests conducted for 6NET the University of Southampton reported an implementation error in Linux SSM, both IPv4 and IPv6 being affected and a fix is confirmed in 2.6.8 -- earlier versions can be used only if the multicast interface is explicitly specified in (S,G) joins. Since MLDv2 is needed only for routers and receivers any recent IPv6 enabled Linux should be suitable for transmitting SSM packets.

Unfortunately GNU libc, the C standard library commonly used with Linux does not yet include the critical structure definitions and constants needed to compile an SSM application written to the recommended version neutral APIs. It is possible for individual Linux vendors or application developers to work around this problem (see 4.6.1 for an example of this) but it makes development of SSM applications on Linux slightly more difficult. Ordinary end users are unaffected by this problem and it should eventually be fixed by the GNU libc team.

3.2.1.2. Enabling SSM

SSM is enabled in all kernels with the necessary MLDv2 code. There does not appear to be any means to disable SSM multicast reception in the Linux kernel.

3.2.2. BSD

3.2.2.1. Status

See section 3.1.2 above.

3.2.2.2. Enabling SSM

See section 3.1.2 above.

3.2.3. Windows

3.2.3.1. Status

There is no MLDv2 for Windows XP at present.

3.2.4. Solaris

At the time of writing, Sun has just released MLDv2 support for a beta image of Solaris 10.

3.2.4.1. Status

The Solaris beta has been verified to support SSM.

3.2.4.2. Enabling SSM

No specific details are available until the Solaris beta NDA is lifted.

3.2.5. Other

At present we are not aware of other platforms that support IPv6 SSM. It would be desirable for PDAs and other platforms to offer support in due course.

4. Porting Applications

4.1. The SSM API

RFC 3678 [SSM-API] specifies an API for specifying multicast source filters. This API is used by SSM applications to tell the operating system which channels (S,G) they want to join. As we'll see later, it also allows other modes of operation. It specifies both an IPv4 specific API and a protocol

independent one. We will here only look at the protocol independent one and suggest that one always uses this. It has both a basic API that allows one to e.g. add new sources one at the time, and an advanced API where one can specify the entire list of sources with just one call. In addition to SSM mode where one joins channels (S,G), it also supports ASM mode. One then joins a group G, and can optionally block certain sources one does not want to receive from. Finally the RFC also has an appendix on an ioctl() based API, but one should avoid that; it is only included for historical purposes. We will now describe the basic and the advanced APIs, and then discuss some possible issues with the API.

This is only a brief explanation of the API. For actually using the API we recommend reading the RFC.

4.1.1. Basic API

There are two structures defined. One is for specifying just a group which is needed for ASM. The other one also allows source to be specified, which is what we need for SSM. The definitions are:

```
#include <netinet/in.h>


struct group_req {
   uint32_t                 gr_interface; /* interface index */
   struct sockaddr_storage gr_group;     /* group address */
};


struct group_source_req {
   uint32_t                 gsr_interface; /* interface index */
  struct sockaddr_storage gsr_group;     /* group address */
   struct sockaddr_storage gsr_source;    /* source address */
};
```

Note that *sockaddr_storage* is used for the addresses. It is large enough to contain either IPv4 or IPv6 addresses.

After filling in the appropriate structure, one can join a group G using:

```
setsockopt(s, slevel, MCAST_JOIN_GROUP, (char *)&greq, sizeof(greq))
```

One can do SSM and join (S,G) using:

```
setsockopt(s, slevel, MCAST_JOIN_SOURCE_GROUP, (char *)&gsreq,
       sizeof(gsreq))
```

And one can block a source using:

```
setsockopt(s, slevel, MCAST_BLOCK_SOURCE, (char *)&gsreq, sizeof(gsreq))
```

For IPv4 and IPv6, *slevel* should *be IPPROTO_IP* and *IPPROTO_IPV6* respectively.

It is not clear whether one needs to join a group before blocking sources or not. It might be reasonable to join the group first, but could also be useful to block a source first to make sure one does not receive data from it after joining the group and before it is blocked.

One can call these repeatedly to join or block additional sources.

There are also corresponding calls for leaving a group, leaving a source and unblocking a source. They are similar to the above but one uses *MCAST_LEAVE_GROUP*, *MCAST_LEAVE_SOURCE_GROUP* and *MCAST_UNBLOCK_SOURCE*. It is also not clear whether all state disappears when leaving a group, or if information about blocked sources is kept.

4.1.2. Advanced API

With the advanced API one can join or block multiple sources in one go.

The advanced API is easier to use, at least if one knows exactly which sources and groups to join or block. One can also probably use both APIs. E.g. to set up a conference where one knows who is present when the program starts, one can use the advanced API. If during the conference one wants to mute a participant, one can use the basic API to just block or unblock that participant.

The advanced API has the following function:

```
#include <netinet/in.h>


int setsourcefilter(int s, uint32_t interface,
                    struct sockaddr *group, socklen_t grouplen,
                    uint32_t fmode, uint_t numsrc,
                    struct sockaddr_storage *slist);
```

One specifies a group, and also a number of sources. The parameter *fmode* should be set to *MCAST_INCLUDE* or *MCAST_EXCLUDE* and determines whether one does SSM and lists the sources to join (include), or if one does ASM and lists the source to block (exclude).

There is also a corresponding function for *getsourcefilter():*

```
int getsourcefilter(int s, uint32_t interface,
                    struct sockaddr *group, socklen_t grouplen,
                    uint32_t fmode, uint_t *numsrc,
                    struct sockaddr_storage *slist);
```

To use it one fills in everything except for *fmode* and *slist* one only specify pointers. These will be filled in with the appropriate values.

numsrc should be the number of sources that can fit into the *slist* array. When the call returns it will contain the number of sources in the filter. Note that the numsrc value might be larger than what *slist* has room for. The application should check for this, and if it needs to know all the sources it should if necessary repeat the call with a larger slist.

4.2. Mad-Flute

MAD/TUT is a project [MAD] at Tampere University of Technology in which a team has been implementing protocols for reliable content delivery over multicast networks. This work builds on the experimental Asynchronous Layered Coding (ALC) defined in RFC 3450, which in turn depends on protocols and methods published as RFCs 3451, 3452, 3453 and 3695. The new protocol introduced in the MAD-FLUTE project is FLUTE - File Delivery over Unidirectional Transport [FLUTE], which is at Internet Draft status at the time of this report. In these protocols reliable delivery is ensured despite the lack of any back channel by the use of forward error correction.

The MAD-FLUTE implementation is a piece of portable software written in C and distributed under the GNU General Public License. It supports both IPv4 and IPv6 operation and works on a many operating systems including Microsoft Windows and a variety of Unix-like systems. It streams one or more files from a sender to either a single receiver over unicast UDP or to any number of receivers using multicast UDP. A FLUTE sender can specify an XML catalogue of files to be sent, or use a simple wildcard expression. The transfer rate and error correction can be adjusted to suit the network environment in which it is used. The latest versions of this software also implement a new Internet Draft "SDP Descriptors for FLUTE" [FLUTE-SDP] for session discovery.

As part of 6NET the University of Southampton added support for IPv6 SSM to MAD-FLUTE (see 4.6.1) and some ad hoc tests were conducted both internally and with other 6NET partners. One important factor determined from this testing was that it was important to adjust the maximum packet size in addition to altering the maximum transmit rate when recipients could be using tunnels or other narrow links. Packets which are too large are discarded by the multicast router rather than fragmented, and there is no back channel to report the problem.

4.3. mcastsend and mcastreceive

These tools are a standard part of FreeBSD, and useful for basic multicast testing.

4.4. vic and rat

There are SSM versions of the vic and rat conferencing tools, available from the University of Strasbourg.

4.5. DVTS

There is an SSM patch for dvts1.0a available: http://www-sop.inria.fr/planete/Hitoshi.Asaeda/dvts/

4.6. SSM enabling unicast applications

To port a uni-directional UDP application to multicast (ASM and SSM) you need to do the following. On the sending side you will need to set hop limit as follows:

```
int hops=30;
setsockopt(fd, IPPROTO_IPV6, IPV6_MULTICAST_HOPS, (char *)&hops,
           sizeof(hops));
```

 Except for that you send UDP as usual. Without the above, the hop limit for multicast will be just 1. On the receiving side the only difference is joining G or (S,G). For SSM, you will need this:

Assume that source and group address are in variables of type "char *" named "source" and "group".

```
int e;
struct addrinfo hints, *res;
struct group_source_req gsreq;

/* Should be possible to use "0" for interface or take interface
   name as argument */
gsreq.gsr_interface = if_nametoindex("eth0");

if ((e = getaddrinfo(source, NULL, &hints, &res))) {
    fprintf(stderr, "getaddrinfo failed: %s\n", gai_strerror(e));
    return -1;
}
/* We only use first value, a name might resolve to multiple */
memcpy(&gsreq.gsr_source, res->ai_addr, res->ai_addrlen);
freeaddrinfo(res);

if ((e = getaddrinfo(group, NULL, &hints, &res))) {
    fprintf(stderr, "getaddrinfo failed: %s\n", gai_strerror(e));
    return -1;
}
/* We only use first value, a name might resolve to multiple */
memcpy(&gsreq.gsr_group, res->ai_addr, res->ai_addrlen);
freeaddrinfo(res);
```

```
        /* Assume IPv6 socket here, for IPv4 use IPPROTO_IP */
        setsockopt(s, IPPROTO_IPV6, MCAST_JOIN_SOURCE_GROUP,
                   (char *)&gsreq, sizeof(gsreq));
```

In addition one might want to bind the socket to the group address, but this is not really necessary.

4.7. Porting ASM applications to SSM

4.7.1.   Experience in porting Mad-Flute to support SSM

The University of Southampton undertook the work of adding IPv6 SSM support to MAD-FLUTE on Linux. For portability reasons the API set out in RFC 3678 "Socket Interface Extensions for Multicast Source Filters" [SSM-API] was used in the new code sections. MAD-FLUTE itself is modular, and already had IPv4 SSM support so only a single module was altered at first, the library which implements ALC. Only approximately 100 lines of code were written or changed at this point.

As mentioned above, the GNU C library provided with Linux does not include the structures and data constants described in RFC 3678 "Socket Interface Extensions for Multicast Source Filters". Similar structures are defined by the Linux kernel, but it would be a mistake to try to use these from an ordinary user space application like MAD-FLUTE. These internal structures are subject to change at any time without notice by the kernel developers, meaning that on future Linux systems the software could malfunction or fail to compile at all.

On the other hand lifting the definitions directly from the RFC could potentially lead to problems with size, ordering and layout of structures. In fact this was tried, and some critical parameters were re-ordered in the group membership API, resulting in a non-working application.

After experimentation a compromise was reached in which a C header file was written which contains a subset of the definitions from RFC 3678 (just enough to properly implement MAD-FLUTE's requirements) carefully edited to match the current Linux kernel ordering and layout rules. Assuming that the GNU C library eventually adopts either the original kernel structures (thereby blessing them as a userspace API) or takes a similar path to the one used here and invents its own equivalent structures with the same order and layout this solution will remain source & binary compatible indefinitely.

Once the modified MAD-FLUTE would compile and run another problem was identified. The kernel multicast API would reject all attempts to join a (S,G) pair with a failure code indicating that an invalid network interface had been selected. Upon examining the source code for SSM in the Linux kernel it was apparent that the default interface (zero) described in the socket API was being treated as a specific interface in the code. This bug was reported to the maintainer for IPv6 multicast in Linux, and a fix was forwarded for inclusion in the next revision of the kernel. Meanwhile as a temporary fix MAD-FLUTE was modified to choose the primary ethernet interface if present instead of relying on the default.

With this change MAD-FLUTE became a fully functional SSM IPv6 application on Linux. In theory changes needed for other operating systems should be very minimal if they implement RFC 3678. So far no attempt has been made to see if this is true in practice.

All the changes described, together with some minor bug fixes were contributed to MAD/TUT and were integrated into the official distribution starting with MAD-FLUTE 1.0. Later versions fixed some further bugs and provided a runtime option to choose the multicast interface for SSM group joins, to replace the temporary solution described earlier. To facilitate testing by 6NET and others the MAD-FLUTE software is packaged up for easy installation on suitable Linux systems by Southampton.

4.7.2. Architectural issues

Many ASM applications use RTP together with RTCP feedback from receivers. The RTCP feedback is sent to the group so that all receivers learn of each other. Both the identity of other receivers and information on reception quality. Also how much bandwidth a receiver uses to send reports is calculated based on the total bandwidth used for reports from other receivers. The more receivers there are, the less bandwidth should each one use. Doing this with SSM is difficult. [RTCPSSM] describes how one can do RTCP with SSM.

4.7.3. Source discovery

For the traditional multicast, ASM, where one simply joins a group, one has SAP (Session Announcement Protocol) for announcing available multicast sessions. For global scope one has one specific group, one for IPv4 and one for IPv6, for announcements. Thus an application can simply always join this group, and see what global sessions are available. For IPv4 with the help of MSDP, there are many RPs for this group, and there is no single point of failure, and there is no particular entity responsible for the service.

In the absence of an ASM service one should try to come up with some replacement for SAP. Perhaps the most naïve way of doing SAP with SSM is to have a single (S,G) instead of a single group G providing the service, and unicasting announcements to S which then resends them as multicast. Such an approach would not scale though, at least not globally. To scale one would at least need some sort of distributed architecture. It is not clear how this should look.

What one usually does for SSM is that one simply lists the session information on a web page, e.g. as with SDP. Different session owners or providers can then list their own sessions on their web pages.

There are cases though where one wants to find all sessions providing a specific content. One may not know in advance what the possible providers are and what web pages to look at. One could imagine electronic programme guides on the web though where one can go to one place and find sessions from many different providers, and also have the necessary search capabilities. But again, it probably does not scale to have all sessions listed like this in one place.

How best to do this is an open problem and we're not trying to solve it here. It would be a great benefit to SSM to find a solution to this.

5. SSM Source Discovery Protocol (SSMSDP)

Source Specific Multicast does not provide an in-band source discovery service as ASM does. Indeed, in the ASM protocol suite composed by PIM-SM and MLDv1, a multicast group has the

same semantic as LAN-style/Ethernet multicasting. Any host can send toward the group, without prior signalling and/or source discovery. This is not the case with SSM, where source discovery is not in the network, but pushed to the end hosts. Then, for multi-source applications requiring an automatic source discovery service, SSM is not usable alone. At ULP, we proposed and implemented a solution called SSMSDP which stands for Source Specific Multicast Source Discovery Protocol.

This solution aims at being as general as possible, to provide an easy transition for applications based on the Any Source Multicast service model.

We describe briefly the proposition and the implementation in section 5.1. In section 5.2, we describe a tool (the *ssmsdpifier*) based on this implementation which allows users to launch ASM dependant applications over SSM only networks without patching or/and recompiling.

In section 5.3, we give examples of tested applications with the *ssmsdpifier*. The first one is *vic* a well known multi-party video conference tool and the second one is the *beacon*, a monitoring tool, based on a well known ASM dependant protocol: RTP/RTCP.

5.1. SSMSDP overview

5.1.1. Architecture and protocol overview

Each emulated ASM group is identified by a **control channel** (C, G) where C is the address of a **controller**. The controller can be implemented in any host, for example on the host of the group creator. The control channel is announced by exterior means, in the sane way as an ASM group address. Hosts wanting to participate in the emulated ASM group join the control channel. A source S_i regularly unicasts a source announcement (S_i, G_i) to C. C regularly sends list of active sources on channel (C,G). Receivers receive source announcements on (C,G) and join each (S_i,G_i).

The SSMSDP protocol uses UDP to encapsulate its messages. Therefore, to announce a session, it is only necessary to advertise this channel and the UDP destination port used (a default port can be used). Receivers only have to join the SSM control channel and bind it to obtain all informations necessary to take part in the session. Source only have to announce themselves to C, to be considered as active in the SSMSDP session. The figure below illustrates this mechanism.
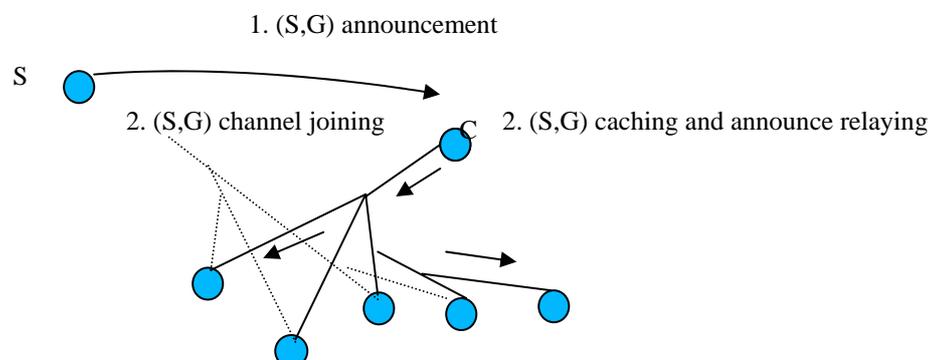
At the application level, it can perform various operations for this session, like advertising new sources, giving the list of existing ones on demand. Having a session control at the application level has the advantage that new functionalities can be easily added, for example session policy

management, on a per application basis. Also, it is possible to choose its location to improve performance. Moreover to manipulate UDP socket is very easy and allows quick implementation on the hosts.

It is possible for the same controller to have authority over several sessions. The controller uses a single UDP socket listening on an announced port and demultiplex source announcement messages based on the G identifier. Several concurrent controllers could run on the same host provided that they listen to different ports to get source announcements.

5.1.2. SSMSDP API implementation

SSMSDP is implemented on top of the SSM/MLDv2 API. It can be seen as a library of functions mapping calls from an ASM application to calls to the SSM API. The SSMSDP API is implemented as C functions in a reusable shared library, libssmsdp available at [] :

*int ssmsdpgetsockopt(int s,int optname,const void *optval)*

*int ssmsdpsetsockopt(int s,int optname,const void *optval)*

Exactly like the POSIX functions *getsockopt* and **setsocktopt**, *ssmsdpgetsockopt* and *ssmsdpsetsockopt* manipulate the SSMSDP options associated with a socket descriptor *s* and immediately return a status code after being called.

These options only exist at the transport (UDP) level as we implemented the protocol on top of UDP. The following table describes the valid option names and associated values.

Option name	*Argument*	*Description*
JOIN	an *ssmsdp_mreq* structure containing the joining interface index and the (S,G) ssmsdp session identifier	Join an ssmsdp session, that is join the control channel and join automatically all sources announced through this channel
CONNECT	an *ssmsdp_mreq* structure containing the controller address and the (S,G) channel to announce	Connect an ssmsdp, that is announce an new channel toward the controller session
CREATE	an *ssmsdp_mreq* structure containing the ssmsdp session identifier (S,G) to create.	Create an ssmsdp session, that is create a process waiting for channel advertisement, cache them an reannounce them in the control channel
LEAVE	an *ssmsdp_mreq* structure containing the leaving interface index and the (S,G) ssmsdp session identifier	Leave an ssmsdp sesssion, that is leave all the previously joined and announced channels, and the control channel
DISCONNECT	an *ssmsdp_mreq* structure containing the controller address and the (S,G) channel to disannounce	Disconnect an ssmsdp session, stop periodic announcements toward the controllers.

Option name	Argument	Description
DESTROY	an *ssmsdp_mreq* structure containing the ssmsdp session identifier (S,G) to destroy.	Destroy a previously created ssmsdp session, that is stop listing and relaying for a particular ssmsdp session identifier

Options *create* and *destroy* are used by the controller only. A standalone application is used to launch a controller. Options *join* and *leave* are used by a session member acting as a receiver while options *connect* and *disconnect* are used by a session member acting as a source.

5.2. The SSMSDPifier

The SSMSDP protocol implementation is based on a library of functions which immediatly return, and execute the protocol in the background. Knowing that, it is easy to derive an application catching the old ASM setsockopt joining/leaving primitives and replacing them with SSMSDP setsockopt primitives. This enables the possibility to port any ASM applications on top of the SSM service without re-compiling or patching. We tested it with two applications, the NLANR (National Laboratory for Applied Network Research) Multicast Beacon and the University College of London Vic. Both worked with no problems, as expected.

The *ssmsdpifier* is this application. Available only for Unix-style systems at this moment, it is based on shared library overloading (to catch asm calls). The ssmsdpifier has the following synopsis :


```
ssmsdpifier [-d][-i joining_iface] -c controller_address progname [ prog_parameters ]

 [-d]   debug mode : print useful informations for debugging.

 [-i joining_iface] Specify the name of the interface used to join  multicast  channels.

 -c controller_address

     Specify  the name (or address) of the controller who is acting as a relay for

     source announcement signalisation.  It acts  as  an  applicative

     rendez vous point.
```


progname [prog_parameters] is the name [and the parameters]of the program to ssmsdpify. If we suppose that the application is running on a host with an IPv6 address I, and if G belongs to the SSM address range, the ssmsdpifier will join the control channel (controller_address, G) in the background, bind on the port P and wait for source announcements from the control channel. These source announcements are sent in the background by libssmsdp toward the controller_address on the same port P, announcing data channel (I,G).

For example

   ssmsdpifier -c ff0e::15 vic ff3e::15/1234

launches vic with group address ff3e::15 (an ssm address although this vic is not ssm enabled) on port 1234. The controller is at address ff0e::15. This means that ssmsdp messages will be send to this controller (and a default port of 7777), and the control channel will be (ff0e::15, ff3e::15).

5.3. Applications tested with the ssmsdpifier.

5.3.1. The NLANR multicast beacon

The multicast beacon is an ASM application using RTP/RTCP. Each host participating in the same beacon session regularly sends data to the group, and gather statistics on data received from all other participating hosts (using RTP/RTCP). This allows to get a matrix of statistics between each host pair (reachability, loss, RTT, ...). This type of tool is very useful when testing multicast deployment in the network. Having this tool run with ssmsdpifier has two advantages :

- verify that ssmsdp (and the ssmsdpifier) are working correctly

- check bidirectionnal SSM connectivity between all pairs of hosts before testing more complex ssm applications.

5.3.2. Vic

The vic conferencing tool has also been tested.

As a conclusion the SSMSDP protocol and implementation both proved they were able to offer a dynamic SSM channel discovery service, and seamless ASM application porting. Note that the ssmsdpifier can be seen as a transition mechanism (using an ASM application over an SSM enabled network without modification). On the other hand the ssmsdp library can be incorporated in new applications and could benefit from added functionalities such as source control.

6. SSM Testing and Experience

6.1. Testbed networks

In this section we provide notes on the testbeds at the 6NET sites involved in the SSM IPv6 tests.

6.1.1. University of Southampton

IPv6 deployment at the University of Southampton is centered in the School of Electronic and Computer Science (ECS). All of the school's networks are IPv6 enabled, and as part of 6NET almost all (see later) were upgraded to support SSM applications on IPv6 using MLDv2 and PIM-SSM.

External IPv6 connectivity to ECS is provided via the LeNSE Regional Area Network. Unfortunately LeNSE is not yet able to provide native IPv6 multicast routing, so a special purpose IPv6-in-IPv4 tunnel runs from Southampton to JANET for multicast traffic.

A Cisco 7203 VXR (Ford) supports the native IPv6 and the multicast tunnel at the ECS endpoint, as well as services such as 6to4 which are useful from nearby networks which have not yet been IPv6 enabled. This router is not used in the native IPv4 networking for the school. It is connected only to the external networks and to the BSD router serving as an IPv6 firewall.

Ford currently runs IOS version 12.3(7)T. The support for SSM in this software version seems to be excellent, although the MLDv2 support was largely untested at Southampton as no hosts are directly connected to this router. Configuration changes to support SSM consisted mostly of

enabling the appropriate MLDv2 and PIM features, and of configuring the router for MBGP in order to receive from JANET routes suitable for SSM Reverse Path Forwarding.

ECS uses PC hardware running FreeBSD for internal IPv6 routing, in parallel with the existing IPv4 routing infrastructure. Because 802.1Q VLAN tagging is already in use in ECS most of the networks are presented as a single physical gigabit Ethernet link which is plugged into one of the FreeBSD routers (Dent). Another of the BSD routers is dedicated to IPv6 router experiments, such as tunnel brokers, one is used for an ongoing Internet radio project (Trillian) and one is used as a firewall (Zaphod).

Zaphod is connected to, and routes between the other three BSD routers and Ford, as well as the ECS IPv6 wireless DMZ which contains all unauthenticated IPv6 users of the 802.11B network. The DMZ has no other physical connection to the IPv6 network for security reasons.

Because the FreeBSD routers were already installed and working, and to minimise disruption to other users the SSM setup in ECS ideally needed to work on unmodified FreeBSD. To this end the patched PIM6SD described above was developed. This has now been running for many weeks on three FreeBSD machines without any further problems in the daemon itself (but see below). Configuration for SSM is minimal, just one line per interface enabling MLDv2 support.

6.1.2. UNINETT

We have mainly done tests with the MAD FLUTE application which are described here. We have also run a beacon for SSM using the normal NLANR beacon written in Perl together with the ssmsdpifier from University of Strasbourg. The topology for the network looks like this:

```
source - UNINETT routers - 6NET routers - NRN/university routers - receivers
          |
       receiver
```

In the UNINETT and 6NET network all the routers involved in the tests are Cisco routers. In the NRN and university networks there are other routers also. This includes at least FreeBSD but there may be others as well. The source and receivers are mostly Linux. It is not known whether there also are non-Linux receivers. In fact it's not known who all the receivers are. The tests, where UNINETT source MAD-FLUTE data, have been announced widely, and with multicast there is no easy way to determine who the receivers are or even the number of receivers.

6.1.3. Strasbourg

The SSM deployment status in Strasbourg has a very simple setup composed of a single Cisco 2600 access router, running MLDv2 and PIM-SM. This router gets m6bone connectivity through a MBGP tunnel toward RENATER. The configuration of this router was straightforward as it only required to enable PIM on the interface.

On the ULP side, several hosts (*BSD and Linux) have been deployed with MLDv2 support. A multicast beacon is running on top of ssmsdp (see above). The 6NET partners can check ssm connectivity by deploying a similar beacon.

6.1.4. PSNC

The PSNC test network consists of four PCs and one hardware router Cisco 7200 series (*css7*). One of these PCs (*tarragon*) also acts as a software router. The main topology of the network is shown in Figure 6-1. For each test the network configuration was a little bit different and all changes are noted below.
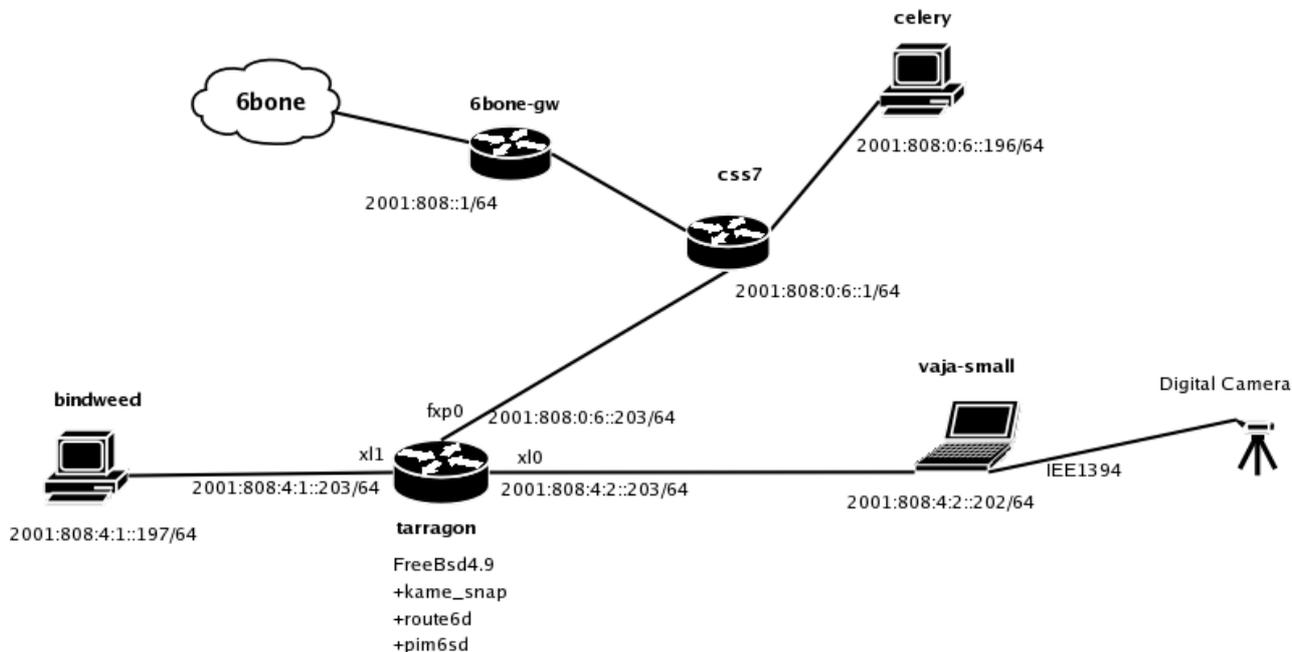
Figure 6-1: PSNC SSM topology

- The PC computer with FreeBSD 4.9-RELEASE was used as a router. It supported MLDv2 protocol with multicast daemon *pim6sd* taken from the KAME project. In order to run *pim6sd the* installation of KAME snap and the compilation of the kernel sources were necessary. Two interfaces with global IPv6 addresses and correctly configured unicast routing were also needed.

The configuration file of the FreeBSD router was as follows:

```
/etc/rc.conf

    ipv6_enable="YES"
    ipv6_gateway_enable="YES"
    ipv6_router_enable="YES"
    ipv6_router="/usr/sbin/v6/route6d"
    mroute6d_enable="YES"
    mroute6d_program="/usr/local/v6/sbin/pim6sd"
```

The configuration file for *pim6sd*:

```
/etc/pim6sd.conf
```

```
phyint xl0 mld_version any;
phyint xl1 mld_version any;
log all;
```

The above configuration enables both versions of MLD (v1 and v2) for two interfaces (xl0 and xl1) and indicates that all events will be logged in */var/log/pim6sd.log*.

On the other hosts the following operating systems were installed:

- Linux RedHat 9.0, kernel 2.6.6 (celery)
  Linux kernels support MLDv2 from version 2.4.22, but in all versions below 2.6.5 the MLDv2 ICMPv6 value is wrong. Beginning from the version 2.6.6 this value has been corrected (*IANA*-assigned value is **143**).

- FreeBSD 4.9-RELEASE, kame-20040712-freebsd49-snap.tgz (*bindweed*)
  To enable MLDv2 on FreeBSD host installation of the KAME snap and recompilation of the kernel are required. Before compilation the proper line in the default KAME kernel configuration file (*kame/freebsd/sys/i386/conf/GENERIC.KAME*) should be uncommented.

- Windows XP and FedoraCore (*vaja_small*)
  Windows does not support MLDv2 but it was not necessary because it was being used only as a source of transmission from a digital camera. A streaming application DVTS 1.01 was used. (This is an important point to note – a *source* does not need specific support since it only needs to send to the appropriate SSM IPv6 group address).

6.1.5. UCL

The SSM deployment at UCL consists of a Cisco 7606VXR access router, running MLDv2 and PIM-SM (Running IOS version: 12.3(20030519:223908) iwu-geo_t_pi2_itd2.eff.May20). This router is connected natively to 6NET via the ULCC/UKERNA PoP.

On the hosts side UCL has deployed a number of Linux-2.6.8 based machines with MLDv2 support.

Successful tests have been performed with UNINETT using the MAD-FLUTE application. The test transmissions on SSM and ASM were successfully accessed using MAD-FLUTE-1.0 under a Gentto Linux-2.6.7 machine.

6.1.6. GRNET

The GRNET test platform is Intel D850EMV2 MB / Intel P4 @ 2.8 GHz CPU / 1 GByte RAM running Fedora Linux Core 2. The test application was FLUTE which compiled without any problems. GRNET participated on two trials, namely scheduled transmission and continuous transmission (all day long). The results for the scheduled tests and continuous tests showed all files received successfully.

GRNET was unable to deploy an SSM beacon due to a PERL installation and configuration problem (some PERL modules missing from the installation).

6.2. MLDv2 Host API testing

6.2.1. History

A first implementation of IGMPv3 and MLDv2 has been realized by the LIP6 laboratory under FreeBSD 4.3 using the IPv6 code integrated by the *kame* project. This implementation is stable and its development has stopped. The second implementation in which ULP was interested, is the one currently carried out by the INRIA and more precisely by Hitoshi Asaeda. This API is developed under NetBSD-current, with the porting to other systems being realized by *kame*. This implementation is based on [draft-ietf-magma-msf-api].

6.2.2. Functionalities offered

This API provides not only the functionalities offered by IGMPv2 and MLDv1 but also these from new versions, namely the filtering of sources. Thanks to the functions of this API, it is possible to put one or more sources in a filter, which can be in INCLUDE or in EXCLUDE mode, switch from one mode to another, in ASM or in SSM.

An interesting feature, is that this API provides functions which are independent from the IP version. This increases the portability of applications developped with it.

The functionalities offered can be delta-based ones, which gives the possibility to add or remove one source at the same time from a filter, in ASM or in SSM thanks to the setsockopt function, or full-state ones, which permits to give a full filter composed of a list of zero or more sources in the filter mode to apply (this is used for switching from a mode to another) with ioctl function.

6.2.3. Conformity tests

As this API is still in development, ULP has written a program which uses it, in order to check its behaviour before integrating it in other applications. This program, called *test-msf* performs all the operations described in the protocol specifications. It has been developed in C language. IGMPv3 and MLDv2 implementations are based on a standard and will be implemented under most operating systems. ULP designed *test-msf* to run under many OS as possible.

The tests that ULP has done are inspired by the behaviour described in the specifications. We tested several transitions. Theses transitions are given in the following table :

Previous setsockopt state	*Following setsockopt state*	*Return*
MCAST_JOIN_GROUP	MCAST_JOIN_GROUP	EADDRINUSE
MCAST_JOIN_GROUP	MCAST_LEAVE_GROUP	NULL
MCAST_JOIN_GROUP	MCAST_JOIN_SOURCE_GOUP	EINVAL
MCAST_JOIN_GROUP	MCAST_LEAVE_SOURCE_GROUP	EINVAL
MCAST_JOIN_GROUP	MCAST_BLOCK_SOURCE	NULL
MCAST_JOIN_SOURCE_GROUPE	MCAST_JOIN_GROUP	EADDRINUSE

Previous setsockopt state	*Following setsockopt state*	*Return*
MCAST_JOIN_SOURCE_GROUPE	MCAST_LEAVE_GROUP	NULL
MCAST_JOIN_SOURCE_GROUPE	MCAST_JOIN_SOURCE_GROUP	EADDRNOTAVAIL if address already filtered
MCAST_JOIN_SOURCE_GROUPE	MCAST_JOIN_SOURCE_GROUP	else NULL
MCAST_JOIN_SOURCE_GROUPE	MCAST_LEAVE_SOURCE_GROUP	EADDRNOTAVAIL if address already filtered
MCAST_JOIN_SOURCE_GROUPE	MCAST_LEAVE_SOURCE_GROUP	else NULL
MCAST_JOIN_SOURCE_GROUPE	MCAST_BLOCK_SOURCE	EINVAL
MCAST_JOIN_SOURCE_GROUPE	MCAST_UNBLOCK_SOURCE	EINVAL
MCAST_BLOCK_SOURCE	MCAST_JOIN_GROUP	EADDRINUSE
MCAST_BLOCK_SOURCE	MCAST_LEAVE_GROUP	NULL
MCAST_BLOCK_SOURCE	MCAST_JOIN_SOURCE_GROUP	EINVAL
MCAST_BLOCK_SOURCE	MCAST_LEAVE_SOURCE_GROUP	EINVAL
MCAST_BLOCK_SOURCE	MCAST_BLOCK_SOURCE	EADDRNOTAVAIL is address already filtered
MCAST_BLOCK_SOURCE	MCAST_BLOCK_SOURCE	else NULL
MCAST_BLOCK_SOURCE	MCAST_UNBLOCK_SOURCE	EADDRNOTAVAIL is address already filtered
MCAST_BLOCK_SOURCE	MCAST_UNBLOCK_SOURCE	else NULL

6.2.4. Robustness tests

A filter with a large number of sources has been created and submitted to the API, to check how it would respond. The biggest number of source that the ioctl can manage at the same time per socket is 63, both for IPv6 and IPv4 in EXCLUDE and in INCLUDE mode. We checked the API with many consecutive operations. These tests were successful.

6.3. Application testing

The focus of IPv6 SSM testing has been the MAD-FLUTE application, ported by Southampton.

UNINETT has been, and still is, running MAD FLUTE to stream internet drafts using SSM for several months. MAD FLUTE can be found at http://www.atm.tut.fi/mad/.

UNINETT has a mirror of all the IETF's internet drafts. Every night FTP is used to synchronise UNINETT's mirror with the IETF site. After this is done, an XML file is created listing all the new drafts. This XML file is used by the MAD FLUTE application, describing the files to be streamed.

At 11 every morning Central European time (CET/CEST) FLUTE streams the new files. It is run with the following parameters:

```
flute -S -a:IP6 -T:40 -m:ff3e::d3af:1 -s:2001:700:e000:0:204:75ff:fee4:423
b -X:100 -x:1 -l:1280 -f:/tmp/fdt.xml
```

The arguments are as follows: "-S" act as source; "-a" use IPv6; "-T" hop limit 40; "m" send to group ff3e::d3af:1; "-s" specifies the source address to use; "-X" sets FEC ratio, 100% means sending twice the amount of data, allowing 50% of the packets to be lost; "-x" sets the encoding to Reed Solomon; "-l" sets the length to 1280 to make sure there is no fragmentation; "-f" specifies the XML file describing the data set. The FEC ratio can be set much smaller, usually the amount of packet loss is very small.

At the receiving side, one should use something like

```
flute -A -a:IP6 -M -m:ff3e::d3af:1 -s:2001:700:e000:0:204:75ff:fee4:423b
```

The arguments are as follows: "-A" for receive automatically; "a" use IPv6; "-M" for SSM; "-m" for the group; "-s" for the source.

The receiver will when started, run until it receives a transmission and then stop. By starting the receiver every morning before the transmission starts, or simply running it in a loop; one will get all the new files, and one can maintain a mirror of the IETF site. The only issue is that there is no way to remove files that are deleted at the IETF site.

For testing there is also a transmission to the group ff3e::d3af:2. The arguments used are exactly the same. The difference is that this loops. At 10 Central European time (CET/CEST) every morning it starts sending the drafts that were received last night. Once they are sent, it waits for 10 seconds, then it sends the same again. This is repeated continuously until 10 next morning, when it starts over with new files. This is useful because at any time people can test that SSM and their FLUTE application works. If one wants to maintain a mirror of drafts one should use the once daily transmission though.

6.3.1. Methodology and Test suites used

There are two test suites for the MAD-FLUTE data:

- SSM group ff3e::d3af:1, for daily transmission at 11 EST time

- SSM group ff3e::d3af:2, just for testing, the same transmission repeated continuously

Most participants were testing on Linux.

6.3.2. Results

Some examples of test results are discussed here.

6.3.2.1. PSNC

PSNC also tested DVTS with IPv6 and SSM.

Before the tests with *Mad-Flute* and DVTS/XDVSHOW were performed the network configuration had been tested with *dtms & dtmc* tools (for details see: *http://amorpha.man.poznan.pl/ssm*). Two instances of *dtms* server were run on Linux machines (*celery* and *vaja_small).* Both two hosts were sending their packages onto the same multicast address (belonged to the SSM scope: ff3e::4861) and the same port. This transmission was received on host with FreeBSD *(bindweed)* using the *dtmc* tool. Each time the source address was changed, it was able to receive data from different hosts (*vaja_small* or *celery*). According to this experience it was posited that SSM test network works fine and next tests could be started.

In tests with *Mad-Flute* v1.0 application as a source of data the host with Linux was used. Another host (*celery*) with RedHat 9.0, kernel 2.6.6 (SSM enabled) acted as a receiver. It was connected directly to the *xl0* interface on software router (*tarragon*).

- In the first test in the local network one file was being sent continuously from PC with Linux *(bindweed)* on channel (S, G) = (2001:808:4:1::197, ff3e::4861).

  The *Mad-Flute* command:

  ```
  flute -S -a:IP6 -i:2001:808:4:1::197 -m:ff3e::4861 -F:LICENCE.TXT -C
  ```

  And this transmission was being received on host with Linux, kernel 2.6.6 (*celery).*

  The command:

  ```
  flute -A -a:IP6 -m:ff3e::4861 -s:2001:808:4:1::197 -M
  ```

- In the second test only PC with Linux, kernel 2.6.6 (*celery)* was used to receive RFC documents available at channel (S, G) = (2001:700:e000:0:204:75ff:fee4:423b, ff3e::d3af:1)

  The *Mad-Flute* command:

  ```
  flute -A -a:IP6 -m:ff3e::d3af:1 -s:2001:700:e000:0:204:75ff:fee4:423b -M
  ```

The test of the DVTS and XDVSHOW applications were performed as well. For this test host with FreeBSD (*bindweed)* and with WindowsXP *(vaja_small)* were used. The *DVTS 1.01* for Windows without any modifications or patches was employed to send a DV stream over IP. The data from the IEEE1394 digital camera connected to Windows machine (*vaja_small*) were being sent onto SSM multicast address ff3e::4861 and default port 1234.

For receiving transmission the XDVSHOW application with applied *patch* taken from http://www-sop.inria.fr/planete/Hitoshi.Asaeda/dvts/ was installed on host with FreeBSD (*bindweed*). While

testing the application some problems occurred (see next chapter) but finally it was possible to see the picture view from the camera.

The XDVSHOW command used on (*bindweed*) was:

```
xdvshow -6 -g ff3e::4861 -s 2001:808:4:2::202 -M fxp0
```

In both test cases with *Mad-Flute* transmission from the selected channel was received without any additional delay but there were a few situations when we had to wait for the beginning of the transmission. Received files were complete and faultless in all test sessions.

During the XDVSHOW test, the quality of received picture was good but there were some grey squares on receiver host screen. No reason for these defects was found. Possibly there was a problem with a graphics library or with some codec under FreeBSD. Probably the problem was not related to SSM, because the same effect occurred while receiving a typical multicast transmission (not SSM).

During all tests the communication and cooperation between the MLDv2 implementations on various operating systems could be checked and verified. These tests indicated that the FreeBSD router part of the MLDv2 protocol implementation could get along with the host part of protocol under Linux kernel 2.6.6 or later. Both sender side and receiver side packets were correctly interpreted by the router, as we could see in *pim6sd* log file. It was also possible to verify the operational capabilities of the SSM protocol and the *Mad-Flute* application in a large scale network with many of heterogonous equipment (software routers, hardware routers from many vendors).

6.3.2.2. GRNET

Here we reproduce an example MAD-FLUTE log extract, as collected at GRNET from the UNINETT source:

```
FLUTE Receiver in automatic mode


100.00% of object received, TOI: 0 (SBN: 0 ESI: 0 LAYERS=1)


FDT Instance received
FDT updated, new file description(s) added


URI: file://sverresborg.uninett.no/all_id.txt (TOI=1)
URI: file://sverresborg.uninett.no/1id-index.txt (TOI=2)
0.45% of object received, TOI: 2 (SBN: 0 ESI: 0 LAYERS=1)
0.90% of object received, TOI: 2 (SBN: 0 ESI: 1 LAYERS=1)
1.34% of object received, TOI: 2 (SBN: 0 ESI: 2 LAYERS=1)
1.79% of object received, TOI: 2 (SBN: 0 ESI: 3 LAYERS=1)
2.24% of object received, TOI: 2 (SBN: 0 ESI: 4 LAYERS=1)
```

```
2.69% of object received, TOI: 2 (SBN: 0 ESI: 5 LAYERS=1)

3.14% of object received, TOI: 2 (SBN: 0 ESI: 6 LAYERS=1)

3.58% of object received, TOI: 2 (SBN: 0 ESI: 7 LAYERS=1)

4.03% of object received, TOI: 2 (SBN: 0 ESI: 8 LAYERS=1)

4.48% of object received, TOI: 2 (SBN: 0 ESI: 9 LAYERS=1)

[1/23 Source Blocks decoded]

4.93% of object received, TOI: 2 (SBN: 1 ESI: 0 LAYERS=1)

5.37% of object received, TOI: 2 (SBN: 1 ESI: 1 LAYERS=1)

ETC

99.43% of object received, TOI: 2 (SBN: 22 ESI: 6 LAYERS=1)

99.88% of object received, TOI: 2 (SBN: 22 ESI: 7 LAYERS=1)

100.00% of object received, TOI: 2 (SBN: 22 ESI: 8 LAYERS=1)

[23/23 Source Blocks decoded]


File: sverresborg.uninett.no/1id-index.txt received (TOI=2)


100.00% of object received, TOI: 0 (SBN: 0 ESI: 0 LAYERS=1)
```

6.3.3. Problems encountered

Some of the issues have been described above and are not repeated here.

The 6NET work has been undertaken over the whole of the last year. In that time some quite fundamental issues have been resolved, and bugs/improvements fed back to the OS and application (e.g. MAD-FLUTE) developers.

6.3.3.1. ICMP values

A typical example is the inconsistent ICMP value used for MLDv2, which caused interoperability problems. Some systems used 206, but now all systems should use the IANA-allocated 143.

6.3.3.2. Firewalls

Although tests from other networks were successful, there was initially no success at ECS when attempting to receive SSM transmissions on the wireless DMZ network, which is used by visitors and some staff and students with laptops.

The only IPv6 router connected to this network was the firewall router, which runs FreeBSD 5.2-RC1 and uses the built-in pf firewall software. Diagnostics on the router indicated that the PIM daemon was not subscribing to the necessary groups. On further examination it was found that the software was not receiving MLDv2 join messages at all. PIM traffic handled by the same machine worked as expected throughout.

Eventually it was determined that the pf firewall software in this version of FreeBSD seemed to be malfunctioning in such a way that packets with a hop-by-hop header, required in MLDv2 for security reasons, were discarded if the firewall was active. With the firewall temporarily disabled our test laptop was able to receive SSM transmissions.

The firewall rules allowing MLDv2 (and indeed all ICMP6 packets) were examined, and rules to allow all hop-by-hop headers were also tried, but to no avail. The exact cause of this problem remains unsolved.

In separate testing being carried out at nearly the same time a problem was found with Mobile IPv6 on the same network. Again the trouble was tracked to the loss of packets using hop-by-hop headers. Unfortunately at the time of writing it has not yet been possible to upgrade the affected machine to a newer, supported release of FreeBSD to confirm whether the problem persists.

Our recommendation for anyone who experiences this problem when attempting to implement IPv6 SSM would be to place another multicast router which does not use the pf firewall between the hosts and the firewall with the problem. The hosts can send their MLDv2 messages to this new router which will not be affected by the problem.

6.3.3.3. General bugs

PSNC encountered two examples.

Firstly, there was a problem with *pim6sd* because of some bugs in it. In versions older than *kame-20040712-freebsd49-snap.tgz, pim6sd* does not work properly. It cannot communicate with other PIM routers in the multicast network when the MLDv2 option in the kernel configuration file is enabled. As the router part of MLDv2 protocol is implemented within *pim6sd*, it is not necessary to enable this option in the kernel configuration file of a router to run it as IPv6 SSM router.

The second problem refers to the XDVSHOW application or rather a patch for it. Although versions above 26.08.2003 support PAL system, the patch still tries to add such functionality to the application what causes some compilation errors. After additional manual modifications it was compiled without errors or warnings. When this problem was fixed another was encountered, this time a bigger one. *XDVSHOW* generates an incomplete ICMPv6 packet type *Version 2 Multicast Listener Report Message* which does not contain the source address field. The part of source code responsible for sending SSM messages was rewritten and after this modification it worked fine.

6.3.4. Troubleshooting and diagnosis

6.3.4.1. Beacons

One good way to troubleshoot is to run a beacon that tracks SSM connectivity over time. One such Beacon is available from Strasbourg: http://canet.u-strasbg.fr/beacon/v6/

Strasbourg and UNINETT successfully achieved end to end IPv6 SSM connectivity through 6NET by using ssmsdp and the beacon.

It is very important that 6NET participants install a beacon before testing other SSM applications: this gives us an SSM connectivity matrix which is useful for later application deployment. Installation instructions for 6NET were available here:

http://clarinet.u-strasbg.fr/~hoerdt/libssmsdp/?topic=Status

6.3.4.2. On a Linux host

In general the application software itself running on the hosts reports no errors. On the rare occasion that errors are reported they are local problems which can be easily fixed, usually a result of operator error.

For example, attempting to use an SSM receiver on a Linux system too old to have SSM implemented results in ENOPROTOOPT, the option is not supported by the protocol.

For software which is untried, or during porting of an application, it can be helpful to see whether the multicast network socket has been created, and whether the appropriate (S,G) multicast join has been registered by the Linux kernel. The former can be discovered using netstat or lsof, as with unicast sockets. The latter information can be found by examining /proc/net/mcfilter6

Idx Device	Multicast Address	Source Address	INC	EXC
4 eth0	ff3e0000000000000000000d3af0001	20010700e0000000020475fffee4423b	1	0

In this example an application has joined ff3e::d3af:1 specifying the source 2001:700:e000::0:2004:75ff:fee4:423b, and the primary Ethernet interface eth0 has either been chosen by the application or selected as a possible multicast interface by the kernel defaults.

If it is suspected that there is no MLDv2 capable router on the network, or that something prevents the MLDv2 capable router from receiving traffic from the host, it can be tested by pinging the IANA reserved link-local address ff02::16 all-mldv2-routers, and specifying the appropriate interface. All MLDv2 multicast routers on the same link should respond.

Other tools such as tcpdump or ethereal can be used. Ethereal could correctly interpret MLDv2 packages, which was very useful during the XDVSHOW debugging process. During all tests, router logs were analysed regularly.

7. Conclusions and Further Work

The basic operation of the SSM protocol with IPv6 has been tested and verified on a variety of platforms within the project. While there are still some implementation gaps, these look likely to be addressed in due course for the vendors/platforms concerned.

The key result was that SSM was deployed between at least eight sites on 6NET, and these were able to receive the MAD-FLUTE data successfully. The next phase of testing will look at more advances on the application side.

ULP plans to continue to test the MLDv2 API, but under Linux, and to further develop *mflute*, an application that builds upon the FLUTE protocol in a similar way to MAD.

Southampton plans an advanced MP3 jukebox application based around MAD-FLUTE distribution.

There are a number of open areas for study and effort:

- New and enhanced source discovery methods; the receiver needs to discover the multicast source and group to join to receive appropriate data or content.

- Developing – if –feasible – SSM interactive applications (multi-source); here the challenge is abstracting the multiparty negotiations to the application layer (where with ASM the any source architecture handles this issue).

- Wider OS support for MLDv2 (including Windows XP and PDA platforms); this is largely down to vendors to work on.

- Wider application support. As yet there are few IPv6 SSM applications; this is not surprising given the technology is new. However, the validation done in this project is a good signal for further development.

- There are some applications like e.g. DHCPv6 that uses a built in and fixed multicast address. The idea is that you can reach e.g. a service in any network using this fixed address. This only works with ASM. The server application joins a group G listening for queries. Applications that need to find the server, sends multicast query to the group address. You cannot do this with SSM because the server does not know in advance what the source (client) addresses will be. The alternative to ASM may be to use unicast with anycast address for the server.

- Encouraging vendors to implement MLDv2 (and MLDv1) Ethernet switch snooping; again a vendor issue, but one that will be important for IPv6 multicast (of any kind) to be widely adopted in sites.

We will report on future work in these areas in the update to this deliverable by June 2005.

8. Glossary

Term	Description
Anycast-RP	A way of using the same RP-address on several RP-routers to do load-balancing, and also fast fail-over, see RFC 3446.
ASM - Any Source Multicast	ASM is the classical multicast service model as described in RFC 1112 where any host can join a given multicast group G, and any host can send a packet with destination address G, and have it delivered to all members of the group G. The sender does not need to be a member of G. Compare with SSM.
Auto-RP	A dynamic protocol for configuring the group-to-RP mappings in a multicast domain.
BGMP - Border Gateway Multicast Protocol	An inter-domain multicast protocol. For each active multicast group, it builds a shared tree between domains that have senders or receivers for the group. See, RFC 3913.
Bi-directional PIM	Uses shared trees, not only from RP towards receivers like in PIM-SM, but also in the other direction from sources towards the RP. There are no PIM registers. See draft-ietf-pim-bidir-07.txt.
BSR - Bootstrap Router Protocol	A dynamic protocol for configuring the group-to-RP mappings in a multicast domain. See also PIM-SM specification, RFC 2362.
Channel	This is used as a term for the source-group pair (S,G) in SSM; see SSM.
DR - Designated Router	The PIM-SM router on a link that is acting on behalf of the hosts on the link. When a host starts sending multicast, the DR sends register messages to the RP. When there are multiple PIM-SM routers on a link, one of them is elected as the DR. Initially the DR will also send join messages on behalf of the hosts and maintains tree state, but in some cases another PIM router on the link can take over; see last-hop router. See also PIM-SM specification, RFC 2362.
Embedded-RP	The PIM-SM router on a link that is acting on behalf of the hosts on the link. When a host starts sending multicast, the DR sends register messages to the RP. When there are multiple PIM-SM routers on a link, one of them is elected as the DR. Initially the DR will also send join messages on behalf of the hosts and maintains tree state, but in some cases another PIM router on the link can take over; see last-hop router. See also PIM-SM specification, RFC 2362.
IGMP - Internet Group Management Protocol	Protocol used between hosts and multicast routers. It's used by IPv4 hosts to report multicast group membership to routers. The latest version is IGMPv3, see RFC 3376. For source-specific reports, like in SSM, v3 is required. For IPv6, see MLD.
Last-hop router	The PIM-SM router on a link that is responsible for sending join messages on behalf of the hosts and maintaining tree state. This is the last router to forward the packets before they reach the host. This is initially the DR, but with multiple PIM routers on the same link, another router may become the last-hop router. See also PIM-SM specification, RFC 2362.
MBGP - Multi Protocol BGP	This is often used for multicast. One may not always want the same topology for multicast and unicast. Using MBGP one can have BGP peerings exchanging both

	unicast and multicast prefixes independent of each other. See RFC 2858.
MLD - Multicast Listener Discovery	Protocol used between hosts and multicast routers. It's used by IPv6 hosts to report multicast group membership to routers. For MLD, see RFC 2710. For source-specific reports, like in SSM, v2 is required, see RFC 3810. For IPv4, see IGMP.
MSDP - Multicast Source Discovery Protocol	An inter-domain protocol. It connects RP's in different domains, so that information about new sources can be distributed between the RP's. See RFC 3618.
PIM - Protocol Independent Multicast	An inter-domain multicast routing protocol. Called protocol independent because it makes use of the unicast routing table for RPF, but is independent of which unicast routing protocols are used to populate the table. There are several variants, see PIM-DM, PIM-SM and Bi-directional PIM.
PIM-DM - Protocol Independent Multicast - Dense Mode	The dense variant of PIM. Dense means that it does flood-and-prune instead of only forwarding where requested. Compare with PIM-SM.
PIM-SM - Protocol Independent Multicast - Sparse Mode	The sparse variant of PIM. Called sparse because it only forwards where requested. Compare with PIM-DM. See also RFC 2362.
RP - Rendezvous Point	For PIM-SM, this is used for source discovery. New sources registers with the RP, and initially traffic is forwarded on RPT which is rooted at the RP. See also PIM-SM specification, RFC 2362.
RPF - Reverse Path Forwarding	RPF is used to determine where to send join messages, or from whom a packet should arrive. The RPF neighbour for a given address, is computed from the routing tables, and is often the next-hop a unicast packet with that destination address would be forwarded to. PIM uses RPF to find out where to send join messages and also performs a so-called RPF check, discarding data packets arriving on the wrong interface. RPF is used by many multicast protocols and flooding mechanisms, also BSR. See also PIM-SM specification, RFC 2362.
RPT - RP Tree (aka shared tree)	The tree that is rooted at the RP and created by (*,G) joins from last-hop routers. For ASM this tree is used at least initially, last-hop routers may create SPT's and switch to them. See also PIM-SM specification, RFC 2362.
RTCP - RTP Control Protocol	RTCP is used by RTP receivers to report on the quality of the data distribution. It also includes a string identifying the receiver. See RTP specification, RFC 3550.
RTP - Real-time Transport Protocol	RTP provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video. Those services include payload type identification, sequence numbering, timestamping and delivery monitoring. It is often used with UDP multicast transport but can also work with other underlying transports. RFC 3550.
SPT - Shortest Path Tree	This tree is rooted at the source, or rather at the source's DR, and the leaves are last-hop routers and/or the RP. It's built by (S,G)-joins. See also PIM-SM specification, RFC 2362.
SSM - Source Specific Multicast	Instead of joining a group G, hosts join a so-called channel (S,G), and the host will only receive from the source S. A host can join several sources with the same group. The source does not need to be member of the group. See draft-ietf-ssm-arch-06.txt.

9. References

[DEPLOY] IPv6 Multicast Deployment Issues, draft-ietf-mboned-ipv6-multicast-issues-01,
P. Savola, IETF Internet Draft, work in progress, September 2004.

[Embedded-RP] draft-ietf-mboned-embeddedrp-07.txt, Embedding the Rendezvous
Point (RP) Address in an IPv6 Multicast Address. P. Savola, B.
Haberman, (Work in Progress), July 2004.

[FLUTE] RFC 3926 FLUTE - File Delivery over Unidirectional Transport. T.
Paila, M. Luby, R. Lehtonen, V. Roca, R. Walsh. October 2004.

[FLUTE-SDP] SDP Descriptors for FLUTE, draft-mehta-rmt-flute-sdp-01, IETF Internet Draft,
Work in progress, H. Mehta et al, October 2004.

[IGMPv3] RFC 3376, Internet Group Management Protocol, Version 3. B. Cain,
S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan. October 2002.

[MAD] MAD Application, http://www.atm.tut.fi/mad/

[MLDv2] RFC 3810, Multicast Listener Discovery Version 2 (MLDv2) for IPv6.
R. Vida, Ed., L. Costa, Ed.. June 2004.

[MSDP] RFC 3618, Multicast Source Discovery Protocol (MSDP). B. Fenner, Ed.,
D. Meyer, Ed.. October 2003.

[MSFAPI] RFC 3678, Socket Interface Extensions for Multicast Source Filters.
D. Thaler, B. Fenner, B. Quinn. January 2004.

[PIM-SM] RFC 2362, Protocol Independent Multicast-Sparse Mode (PIM-SM):
Protocol Specification. D. Estrin, D. Farinacci, A. Helmy, D.
Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma,

L. Wei. June 1998.

[RTCPSSM] draft-ietf-avt-rtcpssm-07.txt, RTCP Extensions for Single-Source
Multicast Sessions with Unicast Feedback. J. Chesterfield, J. Ott,
E. Schooler, (Work in Progress), July 2004

[SSM-ADDR] RFC3306, Haberman, B., and Thaler, D., "Unicast-prefix-based IPv6
Multicast Addresses", RFC 3306, August 2002.

[SSM-Arch] Source-Specific Multicast for IP, draft-ietf-ssm-arch-06, IETF Internet Draft,
work in progress, H. Holbrook, B. Cain, September 2004.

[SSM-API] Socket Interface Extensions for Multicast Source Filters, RFC3678,
D. Thaler et al, RFC3678, January 2004.

[SSM-Intro] An Overview of Source-Specific Multicast (SSM), RFC3569, S. Bhattacharyya, Ed,
July 2003.

[SSM-Refs] IETF SSM documents at SWITCH,
http://www.switch.ch/network/ipmcast/references.html#ssm

[USING] Using IGMPv3 and MLDv2 For Source-Specific Multicast,
draft-holbrook-idmr-igmpv3-ssm-08, H. Holbrook et al, October 2004,
approved as Informational RFC.