A Source Discovery Protocol for ASM Applications in SSM Networks

Mickaël Hoerdt, Frédéric Beck
Université Louis Pasteur – LSIIT
Boulevard Sébastien Brant
67400 Illkirch, France
{hoerdt,beck}@clarinet.u-strasbg.fr

Damien Magoni
Université Louis Pasteur – LSIIT
Boulevard Sébastien Brant
67400 Illkirch, France
magoni@dpt-info.u-strasbg.fr

Jean-Jacques Pansiot
Université Louis Pasteur – LSIIT
Boulevard Sébastien Brant
67400 Illkirch, France
pansiot@crc.u-strasbg.fr

***Abstract*— The Single Source Multicast (SSM) model has received considerable attention from the research community as it could finally bring a scalable solution for some multicast applications. In the SSM model it is the responsibility of the applications to tell the network what are their sources of interest. As a consequence, the SSM model is well suited for one to many multicast applications (such as TV and radio streaming) because there is only one source to discover and it is supposed to be well known. However there still does not exist an SSM session layer protocol and middleware in order to help Any Source Multicast (ASM) applications to discover the presence or departure of the SSM sources. In this paper, we propose a detailed architecture for such a source discovery session layer protocol. We have implemented our protocol as a middleware called Libemu and freely available as an open source library.**

I. Introduction

Classical multicast multi-sources applications use the Deering's [1] ASM group communication model where anyone, including non-members, can send packets with one local operation to a group identifier G with best effort guarantees that its packets will be transmitted to all group members: the network duplicates packets and manages the dynamics of sources and receivers. With the new Holbrook's [2] SSM model it is the responsibility of applications to tell the network who are their sources of interest. In this paper we propose a detailed architecture and implementation of a source discovery session layer protocol for making ASM applications able to use SSM sources. Among the most important sections of this paper, section IV presents an overview of the problem and our solution to it. In section V we describe the ASM over SSM emulation part of the protocol. Section VI presents an optional source management mechanism to perform some control operations on an applicative group. Finally section VII shows the various states in which a source can be.

II. Previous work

In his thesis [2], Holbrook considered the case of using multi-source applications in SSM network only environment. He defined two schemes, an hybrid approach of them and evaluated them: he proposed a sender advertisement scheme using a control channel and a session relaying scheme using a central node, and argues that both of them have advantages and drawbacks. He measured the startup delay of the sessions and simulated both propositions but did not define an architecture for it. That's why we propose a detailed architecture based on his idea in this paper.

In paper [3] Chesterfield and Schooler propose to modify RTP/RTCP to have it working under SSM environment. It is aimed at large scale multimedia distribution and control and dot not consider the case of medium sized multiparty conferences. Their solution could work under specific applications [4], [5] but reduces the information provided by RTP/RTCP protocol for conference applications and could only work with multicast applications using RTP/RTCP. We try here to define an architecture aimed at multi-source applications, as independent as possible from the application and only providing independent services for an applicative group.

An architecture based on session relaying has been proposed in [6]. They propose to put multiple proxies on the network to improve the deployment of SSM. They do not consider the problem of multi-source in SSM networks.

A more recent work proposes to modify the SSM model to allow multiple source to emit toward a channel [7]. They argue that this require little modifications in the forwarding state entries, improve the multicast state scalability, and eliminate the single point of failure problem present in the solutions proposed in [2]. We believe that their solution will require a very high deployment effort as it require to change the IGMP and PIM-SSM specification, in the hosts and in the routers. This paper describes a protocol which could be used in the current SSM architecture to achieve the source discovery in the existing SSM Network. We have implemented the protocol as a freely available library for the research community [8].

III. Motivation and goals

As a first point, we would like to have a quickly usable architecture, according to the latest network and application research efforts. We would like to be able to use multi-source multicast at the inter-domain Internet scale too. This means using SSM service, because it is simple to maintain for operators and because it is believed to be a viable solution for large scale Internet multicast.

Second, this proposition must be scalable regarding the number of announced sources and fault tolerant regarding the

source dynamics. This means using soft-state retransmission and UDP for control messages.

For application performances and ease of portability, the architecture must be as transparent as possible with respect to the classical ASM model.

On the implementation viewpoint, this architecture must be portable, independent of the IP version stack used and easy to use for an application developer.

IV. Problem overview

In this section, we will describe an architecture that solves the problems cited, and permits the use of channels in multi-sender applications. This architecture is inspired by Hugh W. Holbrook's thesis [2].

A. Problems to solve

We aim to use channels in multi-sender applications, but to do so, we have several problems to solve.

Firstly, in the ASM model described by Deering, a multicast group is only identified by its IP address. But here, we will work with SSM multicast channels, which are described by a tuple (S,G) composed of the unicast IP address of a source S and an IP multicast address G where only S is allowed to put data in. But then, knowing that at the moment, in most of the multi-source multicast applications a multicast group is identified by both its IP multicast address and the port used to send data to, the problem of identifying a multicast group appears.

Secondly, on the one hand, in ASM when a receiver joins a group, he discovers automatically the sources when they send data to the group. In the same way, when a new source appears, the receivers learn it simply by receiving the packets sent. On the other hand, with the SSM model, no such mechanism exists, because we work with channels (S,G) where S is the single source being able to emit on this channel.

Consequently, the receivers must have a way of discovering the active sources and being informed of the arrival of new ones. To achieve these goals, we have two choices as exposed in [2]:

- Session Relaying.
- Sender Advertising.

We will now explain a solution, based on sender advertising.

B. Solution proposed

This solution uses at least a channel per source, and a control channel, used for relaying various administration information and source advertising for the group. This control channel identifies the multicast group, and runs on top of UDP for sending data. Therefore, to announce a group, it is only necessary to advertise this channel and the UDP port used, receivers having only to join the multicast channel and bind it to obtain all information necessary to take part in the session. The source of this control channel will be the controller of the group, and will perform various operations for this group, like advertising new sources, giving the list of the existing ones on demand and optionally making some network group management.

As it is possible for the same controller to have authority for several groups, it has been decided to use a single controller that takes care of multiple groups. This controller uses a single UDP socket listening on an announced port to communicate with the sources or the receivers which want to learn the existing sources. The source advertising messages and other administration messages are sent to all the receivers via the control channel. One control channel corresponds to one ASM group at the network level. This controller takes care on a controller S of all the control channels identified by a channel (S,...). These channels do not necessarily use the same UDP port to sent data to.

This controller is run at the application level of the OSI model, because it takes care of the interaction between the network and the applications used in multicast sessions. The advantage of having this controller running on the application layer, is that it is possible to choose its location so that the performances are the best, and it introduces the possibility of doing group management at the application layer. Moreover to manipulate UDP socket is very easy and allows quick implementation on the hosts.

This paper describes a protocol which could be used between the sources or the receivers and the controller in SSM networks to manage sources.

V. ASM over SSM emulation

In this section, we describe the ASM over SSM emulation part of the protocol, which means the way the receivers learn the active sources for a group at a given time, and the new ones which could appear during the session.

A. Protocol description for source-controller interaction

Now we will describe the interaction between a source for a group, and the controller (that means the controller responsible of source announcing) for this group. We'll only take a look at the source advertising, the group management control is optional and is described in Section VI.

1) With a new source: When a new source wants to announce itself, it must begin by informing the controller. The controller then stores the information about the source in its cache. To achieve robustness and scalability, this announcement is sent periodically with a message composed of the protocol header with an optional SDP payload [9], [10]. We call this message ON. It is sent to the UDP socket of the controller's process. This message must contain all the necessary information for a receiver to join the channel used by the source to send its data to the group. It is composed of the new source channel information and possibly transport or protocol over IP information.

It must be noted that the transport information is only destined to applications and not the protocol described here. In the previous example, the application behavior is the same than the ASM one. It means that UDP sources have to send

to the same UDP announced port and that each receiver have to bind on the same UDP port for data reception.

Once this message ON has been treated, the controller announces the source to the receivers and optionally send an ON_{RESP} to the new source to inform it about various session parameters, depending on the group management policy. To do so, it forwards this message to the whole group via the control channel, if the source is allowed to send data to the group. Therefore, all the receivers learn that this source exists, and are able to join the channel it uses to send data if desired. Figure 1 illustrates this mechanism.

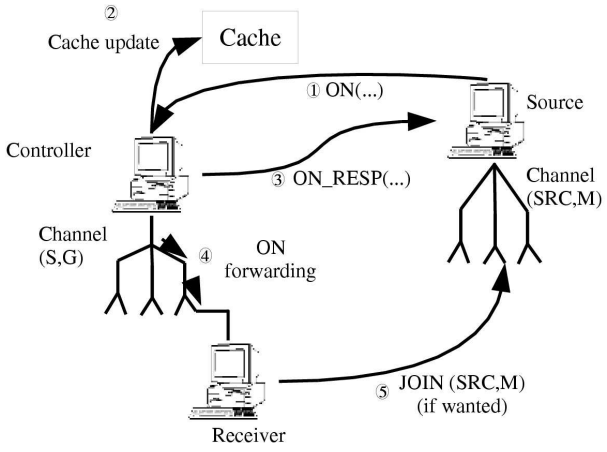

Fig. 1. Interaction Source-Controller for a message ON

2) With an existing source: Periodically, a source must send a message ON to the controller, in order to inform it that it is still active, and the controller forwards this message on its control channel to the receivers. For each source, the controller keeps a timer ON validity which gives the time the source entry in his cache is valid. If the controller does not receive such a message before the timer associated with the source ends, it considers that the source is not active anymore, removes the corresponding entry in his cache and sends a message OFF to inform the receivers that this source no longer exists.

The sources and the receivers also have to maintain timers in order to take care of the validity of the various information and messages. For example, a source has to run a timer for the validity of a message ON in order to take care of the periodical sent of this message.

A source can also explicitly announce that it stops its activity by sending a message OFF to the controller, which removes the corresponding entry in its cache and forwards this message on the corresponding control channel in order to inform the receivers, which then leave this channel. The mechanism is illustrated on figure 2. Note that some upper layer protocols already have a function to advertise the end in a session participation(BYE packets from RTCP protocol). this mechanism may be redundant for some applications but one important thing of our approach is that the control plane is over UDP and the data plane is over IP. This means that any multicast applications working over IP could use this source discovery protocol.

The message OFF is almost identical to the message ON. It is sent to the announced port of the control channel on an UDP socket and is a message composed of this protocol's header with payload containing the identifier of the group and the identifier of the channel used to send data. These two parameters are sufficient to leave the channel corresponding to the source. If the message OFF is lost, the source timer will expire in the controller cache because it is no more updated with the periodic ON messages. As a consequence a OFF message will be sent in the control channel and the receivers will unsubscribe for this source.

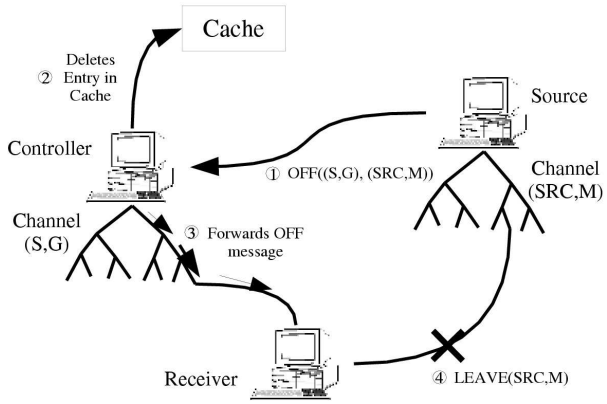

Fig. 2. Interaction Source-Controller for a message OFF

B. Protocol description for receiver-controller interaction

Here we describe the interaction between a receiver and the group's controller. We distinguish the case when a receiver demands information about the sources which are active for the group, and the case when a source has announced itself.

1) Information Request: The controller acts here as a directory of sources. A receiver which wants to learn the list of the existing sources for a group on which the controller has authority, sends him a unicast message on a known port from the receiver. This message is an UDP packet with a payload containing the protocol header and the group identifier, which is the tuple (S,G). We call this message a message $INFO_{REQ}$.

When the controller receives such a message, it takes a look at its cache, and responds to the receiver in unicast with a message $INFO_{RESP}$ giving this list of sources and enough information for the receiver to join the corresponding channels if wanted.

If the controller wants to be able to give these information, it has to maintain a cache containing all these. That means that the cache must contain the group identifier (S,G), if there are more than one controller, the control channels used by these controllers, the dates of beginning and ending of validity of the session, and the source list. That gives the table I.

The source list is composed of zero or several entries. Each entry describes a source and gives the information that a

TABLE I

CONTROLLER'S CACHE INFORMATION

control channel	second control channel	other control channels	beginning	ending	sources

receiver may need to join the channel used for sending data to the group. Such an entry is illustrated on table II.

TABLE II

SOURCE LIST DETAILS

sending channel	port	KICKED	MUTED	coding information	Timer

The flags KICKED and MUTED are used if source management operations are performed, and are set to 0 by default, and the coding information are optional.

This cache is used by the controller to store all the information for a group, but the receivers have to maintain the same cache, but limited to the joined applicative group. This cache is updated thanks to the messages sent on the control channel.

The message $INFO_{RESP}$ is an UDP packet with as payload this protocol's header with a SDP description describing the session and the sources. The SDP part contains the group identifier, the dates of beginning and end of validity of the session, and a list of zero or more channels and ports used by the sources to send data to the group, and possibly more information about these sources which the receivers could need.

When a receiver receives a packet $INFO_{RESP}$, if it has not done so yet, it joins the control channel, and does the same with the channels used by the sources which it has interest in. See Figure 3.

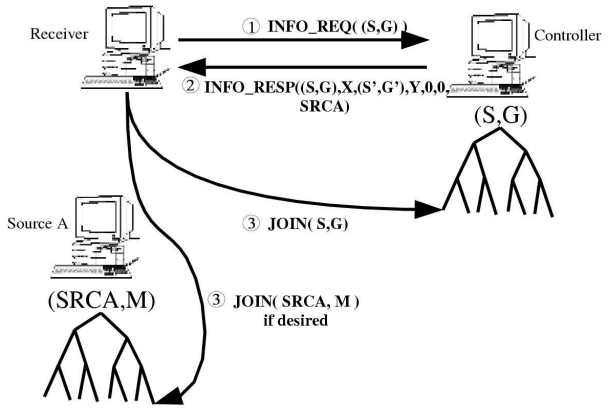

Fig. 3. Interaction Receiver-Controller

2) Source advertising: When a new source announces its activity to the controller. The controller has to inform all the receivers. To do so, it forwards the message ON received from the source on the control channel.

In the same way, when the controller detects the explicit or implicit departure of a source, it forwards or sends, according to the case, a message OFF for this source on the control channel.

VI. SOURCE MANAGEMENT CAPABILITIES

This protocol allows the possibility to perform some control operations on an applicative group. These source management operations are optional. In this section, we give two examples of such operations, the KICK and MUTE mechanisms, inspired by the control operations available in the IRC protocol [11].

A. KICK

The KICK operation consists in forbidding a source to send data to the group. As we are using channels, the controller cannot block the traffic from the source, but it has the possibility to advice the receivers to not receive it. Indeed, if a source is not wanted anymore for a group, the flag KICKED is set in the entry corresponding from the controller's cache, and a message $KICK$ is sent on the control channel for this source. Then, all the receivers learn that the source is not desired anymore and have the possibility to leave the corresponding channel, after having set the flag KICKED in their cache.

As the source is still active, it sends periodically ON messages to the controller, but as long as the flag KICKED is set for this source, the controller does not forward these messages, but instead sends a $KICK$ message.

If the source is authorized to emit again, it is enough to withdraw the KICKED flag and to propagate a message ON for this source on the control channel, the receivers unsetting the KICKED flag in their cache will have the possibility to join again the corresponding channel if it has been left.

This mechanism is transparent for the sources: this operation is done by the receivers which are supposed to leave the channel used by the source.

B. MUTE

This mechanism is less violent than the KICK one. Here the source is not undesirable, but in order to save resources or to maintain session integrity, it is wished that only specified sources transmit data at a given time.

To do so, the controller sends a message $MUTE$ containing the group identifier to the sources that should not send data. When they receive such a message, sent in unicast to their announced port UDP (the address is known from the channel used to send data to), the sources stop sending data on their channels, but keep doing the other things like sending periodical ON messages to the controller.

To authorize the source to emit again, a message $UNMUTE$ containing the group identifier is send to the source.

If a source has been asked to mute, the flag MUTED is set in the corresponding entry in the controller's cache, and this flag is removed if the source is authorized to send data to the group. As long as the flag MUTE is set, the controller periodically sends $MUTE$ message to the source.

This operation is totally transparent for the receivers, unlike the KICK one where they have to perform JOIN and LEAVE actions. But this operation needs the cooperation of the source

which has to stop sending data to the group. That implies that sources also run a process which listens on this port and performs the needed operations if $MUTE$ or $UNMUTE$ messages are received.

C. Timers

To perform all these operations, both the source and the controller have to keep running timers, in order to take care of the validity of $KICK$ and $MUTE$ messages.

If the timer associated to the $MUTE$ message expires, the source decides that it can send data again, removes the MUTE flag and sends again on its data channel.

A receiver has to run a timer giving the validity of a $KICK$ message. If this timer runs out, the KICKED flag is removed and the channel data corresponding to the source is joined (if it was left).

In the same way, the controller has to maintain two timers giving the duration before the next sent of a $KICK$ or $MUTE$ message. The timers used for $KICK$ and $MUTE$ periodical messages have the same value than the timer used for the ON message.

VII. States of the source

In this section, we describe the different states in which a source can be, depending on the action realized thanks to this protocol. We will distinguish the ASM over SSM emulation part and the Control operations.

A. ASM over SSM emulation states

The different states in which a source can be in this case are:

- ACTIVE: the source is considered to be active, notifying it periodically to the controller with ON messages.
- INACTIVE: the source has never been active or has sent a OFF message.

The following state diagram illustrated in figure 4 describes the different states and the transition between them for the ASM over SSM emulation part, which means the source advertising.

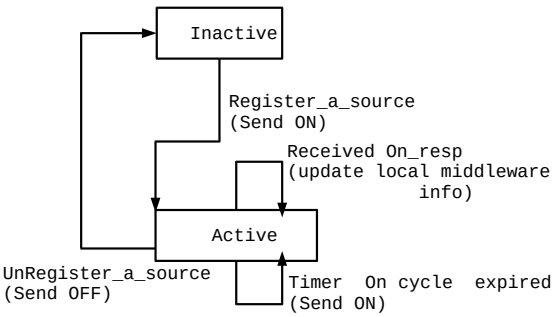

Fig. 4. State diagram for ASM over SSM emulation

B. Source management states

If the optional source management operations are performed, some new states are introduced. The different states in which a source can be in this case are:

- ACTIVE: the source is normally active.
- MUTED: The source has been asked by the controller to stop sending data on its channel.
- KICKED: The source has been KICKED by the controller (but it is still active).
- INACTIVE: The source has announced that it is not active anymore.

The MUTE state is transparent for the receivers, the muted source's channel (multicast tree) is kept. This state is only available for sources. In the same way, the KICKED state has only sense for the receivers, the channel of a kicked source is removed (actually it is virtually removed by the controller's $KICK$ message, the source keeps its data channel and keeps on sending data on it). We can imagine that we send a message $KICK$ to the source to advertise it that it has been kicked from the session, but it is not an obligation as the receivers will not get the data anymore. The INACTIVE state is reached when the receiver has received the OFF message forwarded on the control channel.

The following state diagram illustrated in figure 5 describes the different states and the transition between them for the source management operations.

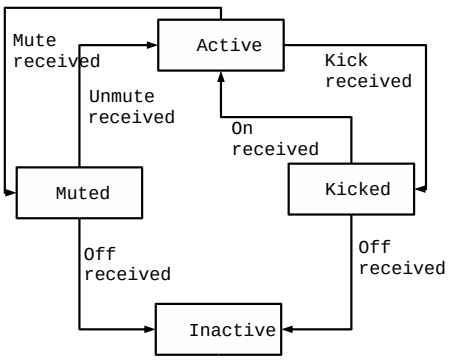

Fig. 5. State diagram for source management

VIII. Breakdowns and consequences

The basic idea to prevent breakdowns is setup a backup controller. The master controller S sends on its control channel periodical ALIVE messages to show that it is always active. This message is a UDP packet with this protocol's header and a SDP description containing the group identifier for which it is the controller.

If the backup controller S' does not receive any ALIVE message from S on (S,G) after the configured time, it considers that S is down and begins sending ALIVE packets and source

advertising. A source which was kicked by the other controller will be announced as active the next time it sends a message ON unless the new one kicks it. In the same way, a source that has been asked to mute, after the timer of the validity of the $MUTE$ message has expired, will begin to send data again. To avoid this situation, it is possible to distribute this by adding a KICK flag in each receiver's cache. Then, if the master controller kicks a source, it announces it with a message $KICK$, which causes this KICK to be set in all receivers, including the second master. This flag would be valid until a ON message for this source is sent. Such a mechanism gives us at the application level the possibility for the application based on this protocol to ask the user if he still wants to receive data from the source even if the controller recommends to kick it, or if he wants to follow the controller's order in case of a $MUTE$ message.

Supposing that the master controller S breaks down, S' detects it and begins assuming this role as already described earlier. A receiver that has already joined the group is a member of both control channels, and then receives the control data on (S',G'), and except the time until S' takes the control, nothing special happens. In the same way, a receiver that wants to join the session joins both control channels, and has the information he needs from S'.

IX. Timers and their default values

The messages ON, $KICK$ and $MUTE$ are sent periodically in order to add robustness to the protocol. Therefore, we have to use several timers to define the validity of the different states and information.

In this section, we describe the different timers used by this protocol and their default value. Some of them are configurable, the other ones depend on each others.

- $ON_{Validity}$: This timer describe the validity of a source entry in the controller's cache. It is the reference used to define most of other timers. If we allow its value to being configurable, we could distribute it to the sources and the receivers thanks to the ON message. Default: 15 seconds.
- $ON_{Forwarding}$: This timer describes the period the controller uses to send ON messages on the control channel. Default: 1/3 * $ON_{Validity}$ (5 seconds).
- ON_{Cycle}: The ON Cycle is the period between two ON messages sent by a source. In order to be robust to one message ON lost, we must have 2 * ON_{Cycle} > $ON_{Validity}$ Default: 1/3 $ON_{Validity}$ (5 seconds).
- ON_{Src}: The ON Src is the time of validity of a ON message for a receiver. Default: $ON_{Validity}$ (15 seconds).
- $KICK$: It is the time of validity of a $KICK$ message sent by the controller, and is used at the receiver's level. Default: $ON_{Validity}$ (15 seconds).
- $MUTE$: It is the time of validity of a $MUTE$ message sent by the controller, and is used at the source's level. Default: $ON_{Validity}$ (15 seconds).
- $ALIVE$: It is the time of validity of an ALIVE message sent by the controller, and is used at the receiver's level. Actually, it only makes sense for the other controllers if we have redundancy in order to add robustness. Default: 2 * $ON_{Validity}$ (30 seconds).
- $ALIVE_{Cycle}$: It is the time between two ALIVE messages sent by the controller. Default: 3/4 * $ON_{Validity}$ (12 seconds).

X. Scalability considerations

An important question is the scalability of this architecture, especially at the controller point. The controller is acting as a sort of rendez-vous point *ala* PIM, and a relay node for control messages between sources and receivers in the emulated ASM group. It is then a central point of failure, sensible to the bandwidth generated by the different announcing sources.

We consider here the problem of bandwidth scalability at the controller in the worst case with the default timers values. What is the amount of required download bandwidth at the controller point in function of the number of synchronized announcing sources?

If we take the messages defined in [12], an ON message which contains one channel will take 158 bytes on the wire, considering that the source is transmitting on Ethernet and IPv6 and that the announced channel is IPv6. With IPv4 it will take 90 bytes considering that the announced channel is IPv4.

In a critical case, if there are N synchronized sources announcing themselves and located at 100 different hosts and that all the messages arrive together at the controller, the mean bandwidth required is a simple linear function, N*158 byte/sfor IPv6 and N*90 byte/s for IPv4 as illustrated in figure 6.

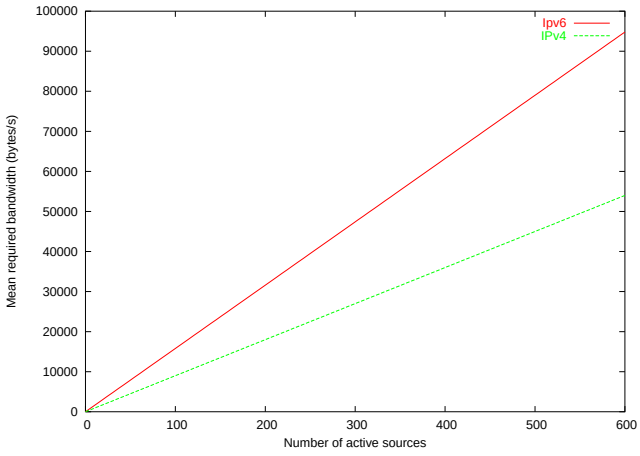

Fig. 6. Required download bandwidth vs number of announcing sources.

The generated bandwidth is not too high as we are considering this case and that applications with more that 100 applicative sources at the same time are very specific and will be certainly based on the ASM network service model if it is finally deployed. Moreover thanks to the acknowledgement mechanism, it is possible to rate limit the sources at the controller. Note that once the messages arrive at the controller,

the Controller has complete control over the bandwidth usage of its control channel.

XI. Security considerations

At this moment, this protocol does not provide security mechanism at all. The only security warrants are based on IP. Because this protocol is based on top of a connectionless protocol (UDP), it's very easy for someone to abuse the security mechanisms offered by IP with techniques like IP spoofing. This is why it is very important on the receiver part to bind on the two part of the (S,G) control channel. According to the SSM service model, and how the SSM multicast tree construction is done at the IP level, it is more difficult to spoof a SSM channel than an IP.

More security mechanisms could be imagined, like using a single session identifier per group which is only distributed between the controllers.

But the security problem is not limited to the controller. We can imagine that a source usurps another one's identity and carries out false advertisements for this one. By doing so, a source can be announced as no more active whereas it still sends data. To avoid this, we could use an appropriate authentication mechanism, which can be done thanks to the acknowledgment messages for example.

XII. Implementation

Our architecture is further defined in an Internet draft available to the research community [12]. In particular, this draft contains the full description of the protocol messages, the format of the packets as well as the definition of all headers and flags. We have implemented our middleware as a library called Libemu that can be used by applications running over Windows, Linux or *BSD operating systems depending on the IP stack version. The library is implemented in C++ and consists in more than 8000 lines of code. It is freely available for download at [8]. It's worth noticing that our library makes use of the GNU Common C++ library (freely available) and needs an host implementation of the multicast protocols IGMPv3 and MLDv2. These protocol implementations are provided by the Kame's project under *BSD and can be found in the latest GNU/Linux Kernel version 2.4.22 or in the new 2.6.x versions.

XIII. Conclusion

Our work aims at providing a multicast library for multi-source applications wanting to connect to Source Specific Multicast only networks. Source Specific Multicast is a new multicast over IP distribution paradigm: by pushing the source selection in the hosts, it is a good candidate for an Internet wide multicast service offer. It is well fitted for TV and radio streaming applications but it does not provide automatic source discovery as the original Any Source Multicast paradigm from Steeve Deering does. Considering multi-source applications with source dynamics and old applications based on the ASM model, a multicast library (usable as a middleware) implementing a source discovery protocol was missing. In this paper we have filled this gap by designing an architecture able to provide such a discovery service. We have defined a new protocol and implemented its corresponding middleware in order to enable ASM applications linked against our library to discover SSM sources. Our next task consists in assessing the performance of our proposal through extensive simulations. We also plan to evaluate our implementation in the environment of the 6NET project and to define a proxy architecture based on this protocol in order to use unmodified ASM applications on SSM networks.

XIV. Acknowledgments

This work was partly sponsored by the *6net* European project under contract number IST-2001-32603.

References

[1] S. Deering, "Multicast routing in a datagram internetwork," Ph.D. dissertation, Standford University, Dec. 1991.

[2] H. Holbrook, "A channel model for multicast," Ph.D. dissertation, Standford University, Aug. 2001.

[3] J. Chesterfield and E. M. Schooler, "An extensible rtcp control framework for large multimedia distributions," *IEEE Symposium on Network Computers and Applications (NCA-03)*, Apr. 2003.

[4] S. McCanne and V. Jacobson, "vic : A flexible framework for packet video," in *ACM Multimedia*, 1995, pp. 511–522. [Online]. Available: citeseer.nj.nec.com/mccanne95vic.html

[5] V. H. an M-A. Sasse and I. Kouvelas, "Successful multiparty audio communication over the internet," *Communications of the ACM*, vol. 41, no. 5, May 1998.

[6] D. Zappala and A. Fabbri, "Using ssm proxies to provide efficient multiple-source multicast delivery," in *IEEE Globecom, Sixth Global Internet Symposium*, Nov. 2001. [Online]. Available: http://citeseer.nj.nec.com/447783.html

[7] K. Sarac, P. Namburi, and K. C. Almeroth, "Ssm extensions: Network layer support for multiple senders in ssm," in *ICCCN*, Dallas, Oct. 2003. [Online]. Available: http://www.utdallas.edu/ ksarac/research/publications/ICCCN03.pdf

[8] F. Beck and M. Hoerd, *Libemu 1.0*, Université Louis Pasteur, http://clarinet.u-strasbg.fr/ hoerdt/libemu/.

[9] M. Handley, C. Perkins, and E. Whelan, "Session announcement protocol," Internet Engineering Task Force, Request For Comments 2974, October 2000.

[10] M. Handley and V. Jacobson, "Sdp: Session description protocol," Internet Engineering Task Force, Request For Comments 2327, April 1998.

[11] J. Oikarinen and D. Reed, "Internet relay chat protocol," Internet Engineering Task Force, Request For Comments 1459, May 1993.

[12] F. Beck, M. Hoerdt, and J.-J. Pansiot, "Source discovery protocol in ssm network," Internet Engineering Task Force," Internet Draft, June 2003.