| | |
|---|---|
| Project Number: | IST-2001-32603 |
| Project Title: | 6net |
| CEC Deliverable Number: | 32603/ULP/DS/5.10/A1 |
| Contractual Date of Delivery to the CEC: | 30 May 2005 |
| Actual Date of Delivery to the CEC: | 16 June 2005 |
| Title of Deliverable: | **Report on using multipoint applications over SSM** |
| Work package contributing to Deliverable: | WP5 |
| Version: | 0.5 |
| Type of Deliverable*: | R |
| Deliverable Security Class**: | PU |
| Editors: | Jean-Jacques Pansiot (ULP) |
| Contributors: | Mickael Hoerdt, Vincent Lucas, Jean-Jacques Pansiot (ULP), Stig Venaas (UoS, Uninett) |
| Reviewers: | Graca Carvalho |

* 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 order to deploy multiparty applications using SSM, we not only need routers running SSM and hosts running MLDv2, but we must also consider some issues: We show that an architecture such as *sas* allows to keep a global session announcement a la SAP using SSM in the inter-domain. We evaluate *ssmsdp* and show that it is a viable architecture for discovering dynamic sources in a multiparty session using SSM. We also evaluate some mechanisms for congestion control in unidirectional multicast application. Finally we discuss some tools such as *ssmping* and *dbeacon* that can help users and administrators when running SSM applications.

**Keywords:** multicast, SSM, source discovery, session announcement, congestion control

## Executive Summary

This document is an update and complement of D5.9. The previous document focused on running SSM on hosts and routers, and porting applications to SSM. In the present document we will focus on some problems that have to be solved if we want users to actually use multicast applications over SSM, especially if SSM becomes the sole inter-domain architecture deployed. We address three main issues, and discuss some tools.

The first issue concerns session announcement. Currently sessions are announced either by offline means (web, mail, news,) or by multicasting so-called session description files using for example the *sdr* utility. However *sdr* uses an ASM group in order to allow anyone to announce its sessions without relying on a centralized server. With SSM this cannot work. Several solutions have been proposed to achieve this functionality using SSM channels. In this document we describe the *sas* architecture developed at UoS and some experiments on the 6net network.

The second issue concerns source discovery inside an SSM-only session. In D5.9 we described the SSMSDP architecture and implementation. In the present document we give the results of some local and inter-domain experiments between several 6NET partners, showing that this solution is scalable to applications with hundreds of sources and sessions using the same controller. Applications (the *dbeacon* and *vic*) have been used with SSMSDP without any modification thanks to *ssmsdpifier*.

The third issue concerns the possible lack of feedback for applications using SSM. In point to point applications, communications are usually bidirectional. Data flows one way, while feedback flows in the opposite direction. Feedback is mainly used for reliability (acknowledgments) and flow control. In multi point applications using ASM, receivers can still send feedback to the group hence to the source, at least with small to medium groups. SSM is well suited for very large groups without feedback. Reliability can be achieved by using FEC such as in *flute*. However we still need flow control/congestion control. This can be achieved using multiple channels of increasing throughput. Receivers dynamically adjust the number of channels they ask for to the current network condition. RLC and FLID-SL (and FLID-DL) have been proposed to do this. We have conducted some experiments using two implementations of flute (Mad and Inria) with several setups (number of sessions, IGMP host tracking, TCP friendliness, RLC or FLID-SL).

Finally, since SSM multicast (even multicast) is not universally guaranteed yet, there is a deep need of tools helping users or network administrators when an SSM multicast application is not working correctly. We present the *dbeacon* and *ssmping* to this end.

# Table of Contents

# 1. Introduction

Source-Specific Multicast (SSM) is a more recent form of IP multicast compared to the "traditional" Any Source Multicast (ASM) model. In particular SSM greatly simplifies or eliminates some problems of ASM (source control in a group, address allocation, discovering and managing Rendezvous Points) and is a good candidate for inter-domain multicast. The 6NET project has been at the forefront of SSM testing and deployment. Deliverable D5.9 described the theory, practice, implementation, testing and issues involved. In this new document we focus on some remaining issues. More specifically we deal with - session announcement in the case where an ASM service a la SAP is not available, - dynamic source discovery in multiparty applications using SSM, - reliability and congestion control when using SSM without feedback channel. We also study some basic tools such as *sssmping* and beacons. These tools allow users or managers of SSM applications to test SSM connectivity across the Internet.

## 2. Announcing Sessions with SSM

Most multicast applications work fine in the inter-domain multicast space using only SSM or Embedded-RP. One particular application or service that is hard to deploy using SSM or Embedded-RP is the Session Announcement Protocol [SAP], RFC 2974. For sessions with global scope (scope E), SAP announcements are sent to a specific global group (FF0E:0:0:0:0:0:2:7FFE). Everyone using SAP sends to or joins this group in order to announce or receive global announcements to or from others. What makes SAP difficult is that it is a totally distributed application with no owner and there should be no centralised infrastructure like an RP (needed for embedded-RP) nor an SSM control channel needed in the SSMSDP proposal.

The tool *sasd* [SASD] has been implemented as a prototype of a solution to this problem. The philosophy of *sasd* is that for applications and users, SAP should be available as today so that changes required at end sites are minimal. In this solution SAP is used as today inside the site, and a site should run its own RP for the global SAP ASM group (this is similar to what happens for IPv4 with MSDP). A sas daemon *sasd* then connects the sites together so that announcements made in one site are available to others. This is done by building a full mesh of SSM channels between participating sites. A *sasd* in one site will pick up its site local announcements from the SAP group and resend them on its SSM channel, and it will forward announcements it picks up from other *sasd*'s SSM channels on the site's SAP group. *sasd* stands for Session Announcement Service Daemon, and is loosely based on the paper "Multicast Session Announcements on top of SSM" by P. Namburi and K. Sarac [ICC04].
Currently *sasd* is based on a full mesh as mentioned above. This is only the first step to test the general principle. If this gets deployed on a large scale, the intention is to further develop *sasd* to build a hierarchy similar to what is described in the paper.

Using a full mesh of SSM channels, or possibly some hierarchy of channels in order to perform distributed (no single point of failure) communications can be useful for many applications. This can be used to emulate ASM. So although *sasd* is for one particular use, it may demonstrate a principle that is of more general use.

The main problem with the full mesh of SSM channels that *sasd* uses, is how to build the mesh without requiring every *sasd* to be configured with the addresses of all the other *sasd*'s. The basic idea with *sasd*, is that it is configured with a few other *sasd* addresses. It will then make itself known to those, and they will tell it about all the other *sasd*'s they know of. Mathematically speaking, it's based on a relation that is symmetric and transitive. That means, if I am related to you, then you are related to me. If I am related to you, and you are related to someone else, then I am also related to that someone else. The mesh created is the transitive closure of this relation. Note that one may end up with several disjoint meshes. A way to avoid this might be to have a few well-known *sasd*'s that everyone specifies.

# 3. Dynamic discovery of sources

## 3.1. Problem statement

In multiparty applications (games, videoconferencing, …) every participant must get data from each other, including new participants arriving after the start of the application. With ASM this is done at the network layer, where a group is identified by a group address and packets sent to this address are received by all receivers no matter the source (using for example a RP and a shared tree with PIM-SM). With SSM, at the network layer groups are replaced by channels, and participants must join a distinct channel for each source they wish to receive. This means that hosts must dynamically discover all the sources of a session they are participating in. This problem has been described in [SSM-DISC]. Each application could solve this problem in a specific way. However a generic solution that can be implemented as a middleware has many advantages : developers of new applications will have a simpler task, and it will be easier to port existing applications .

## 3.2. The SSMSDP architecture

In SSMSDP (SSM Source Discovery Protocol), designed at ULP [SSMSDP] a multiparty session is identified by a so-called control channel (C, G) where C is the unicast address of a controller and G is an SSM address. Every receiver must register to this channel. When a source wishes to send to this session (and later on periodically) it must first send in unicast an ON message to the controller. The controller maintains a cache of all known sources and announces them in the control channel. When a receiver gets a new source announcement it registers to this channel. Since many channels may be bound to the same socket, the application may receive all sources through the same socket as in ASM. When a source stops, it sends an OFF message that is relayed to all receivers through the control channel. The controller may also flush sources after a timeout.

## 3.3. Experiments with SSMSDP and scalability

While SSM plus SSMSDP allow to run multiparty applications over SSM as if an ASM service were available, one may wonder if this architecture scales to very large groups (with many sources) and many sessions: what are the limits for a given controller, and what are the delays introduced by SSMSDP. For this purpose, SSMSDP has been implemented in a library, *libssmsdp* and several experiments, both local and inter-domain have been conducted. What we want to measure is the delay between the time when a source sends an ON message and the time when a receiver has joined the corresponding channel. Below is the basic setup. RPC1, RPC2 and RPC3 are routers (not directly involved in SSMSDP) and H1, H2 and H3 are hosts. In our scenarios, H1 is the source, H2 is the controller and H3 is a receiver. H1 and H3 have joined the control channel. When H1 announces a new source channel, an ON message is sent in unicast to H2 (delay t2), which forwards it in multicast to receivers (delay t2). Therefore we can have an estimate of t1 + t2 when the ON messages gets back to H1. When another receiver (say H3) receives the ON message it joins the new channel. In order to know when this join is complete, we run PIM on H1. The new channel is operational when the join arrives at H1 (after delay of t1 + t2 + t3) where t3 is the delay to construct the new channel. Therefore we can also estimate t3. An interesting point is the ratio between the time to inform the receiver of the new source (t1 + t2) and the time to join the channel (t3). An experiment consists in measuring variation of delays when the number of channels or the number of sessions increases.

### 3.3.1. Experiments with a local test bed

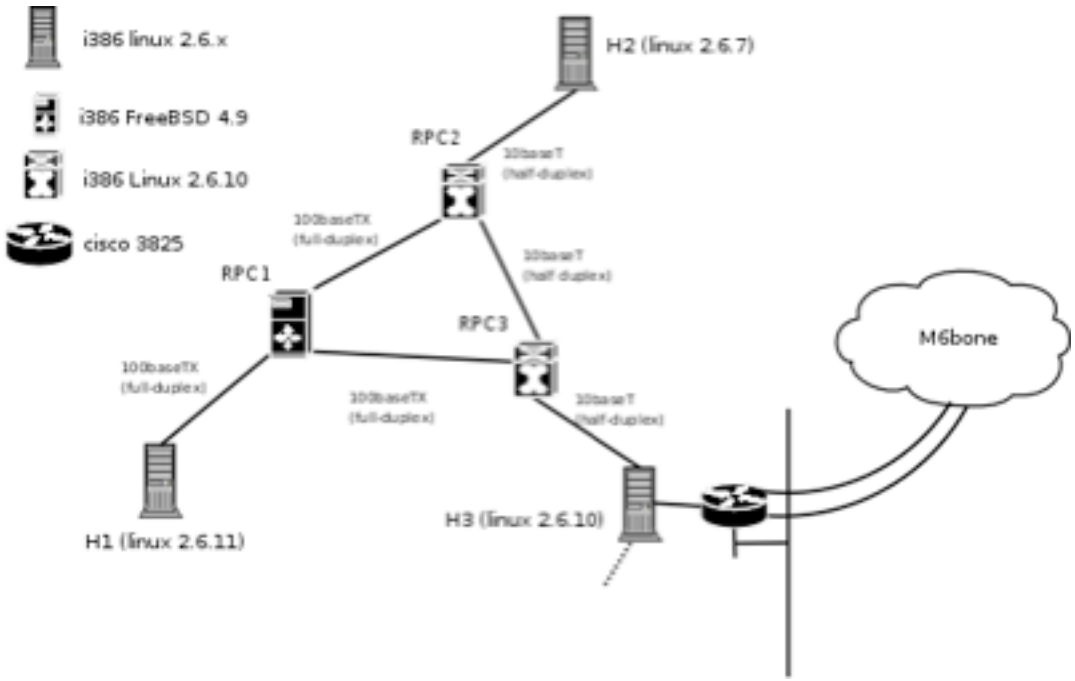



Figure 1. The local setup

In a first experiment we increase the number of channels (data sources) in a single session from 0 to 1500. This is done 10 times, and the mean is given. Figure 2 shows that the delay for source announcement (t1 + t2) is quite stable, and increases slowly with the number of channels (from 2 to 11 ms). In comparison, the delay to construct the PIM branch is higher and much more variable. This seems due to PIM and MLDv2 behaviour.

The second experiment is similar except that we create one by one a large number of sessions in the same controller H2. Figure 3 shows the delays when the number of sessions increases from 1 to 500. Obviously, delays are higher. However since an SSMSDP controller is an application entity on a host, 500 simultaneous sessions is already a big number. A much more detailed account of these experiments can be found in [GLOBECOM05]. Technical details and implementations can be found at [SSMSDP].

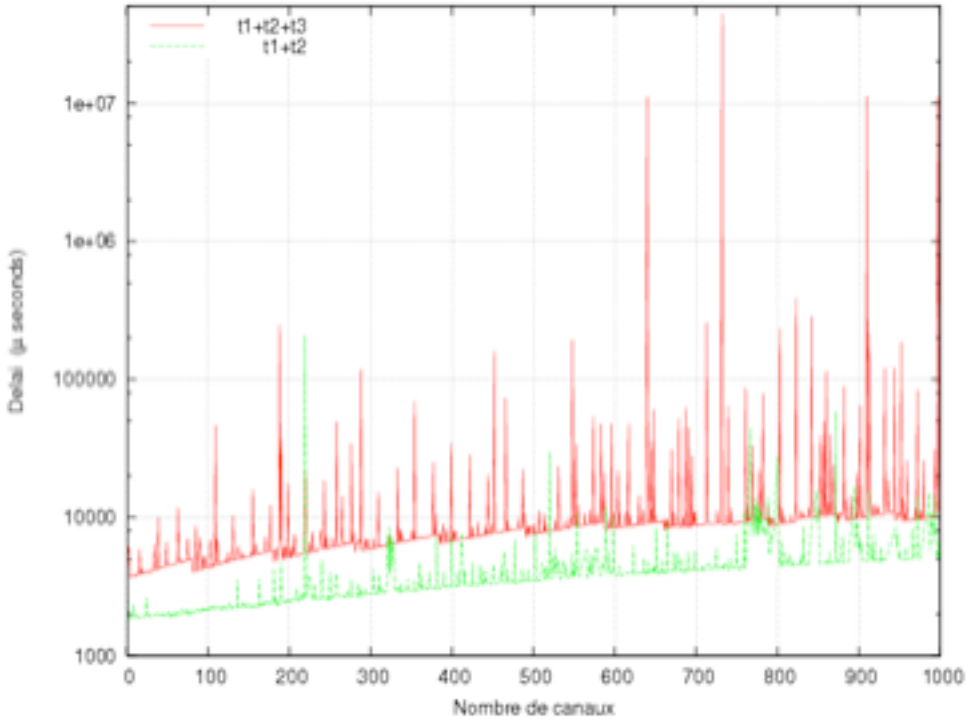



Figure 2 Delay Variation with number of channels

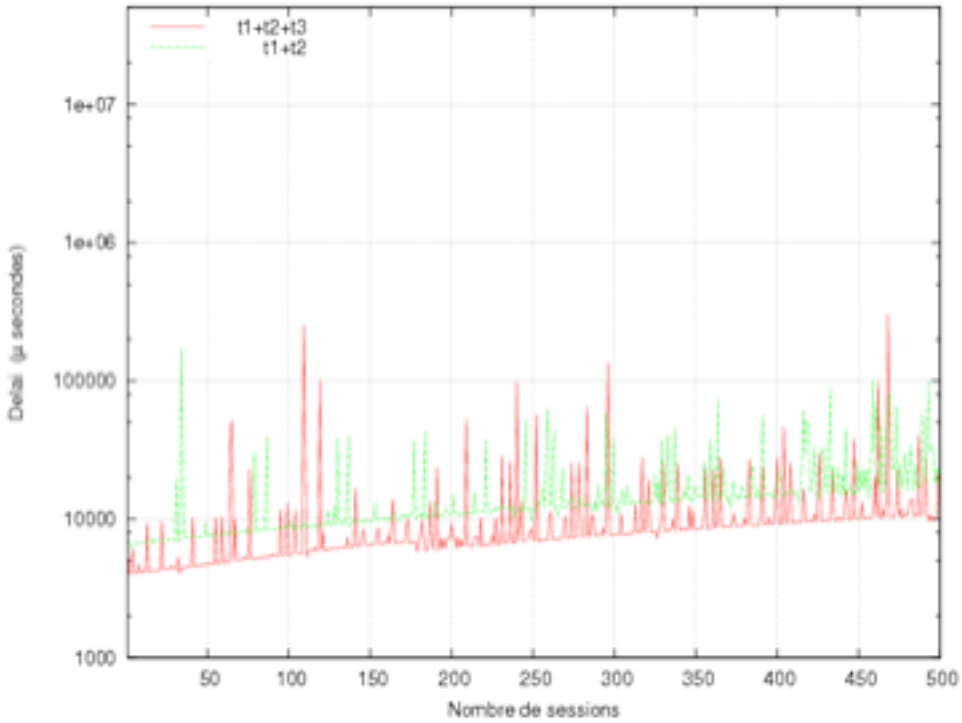



Figure 3. Delay variation with number of session on the same controller

### 3.3.2. European test bed

Obviously delays will heavily depend on network delays between sources, receivers and controllers. We have conducted experiments with two different setups (Figure 4): one triangle between Strasbourg (ULP), Trondheim (Uninett) and Munich (IABG), the second between Strasbourg,

Trondheim and Lisbon (FCCN). These tests use multicast IPv6 over the M6bone (including some tunnels when native multicast is not available). The number of multicast hops is as follows: Strasbourg-Trondheim 13, Trondheim-Munich 13, Munich-Strasbourg 5, Trondheim-Lisbon 10, Lisbon-Strasbourg 8.

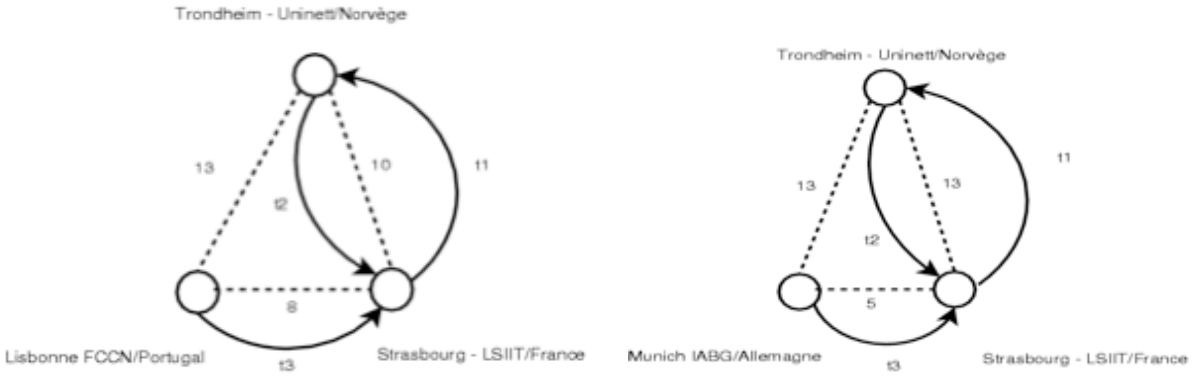



Figure 4. Two European setups with number of multicast hops

Figure 5 gives the delays when the number of channels increases between 1 and 1000, with the controller in Trondheim, the source in Strasbourg and a receiver in Munich. These measures are the mean of 10 trials. One can see that a receiver is informed of a new source in about 50 ms (the main factor is the round trip time at the IP level) but the receiver has joined the new channel after more than 1s. This is due to the time it takes to construct a PIM branch through a number of routers. Also, one can observe that these delays are more or less independent of the number of channels since the overhead on hosts is small compared to networks delays.

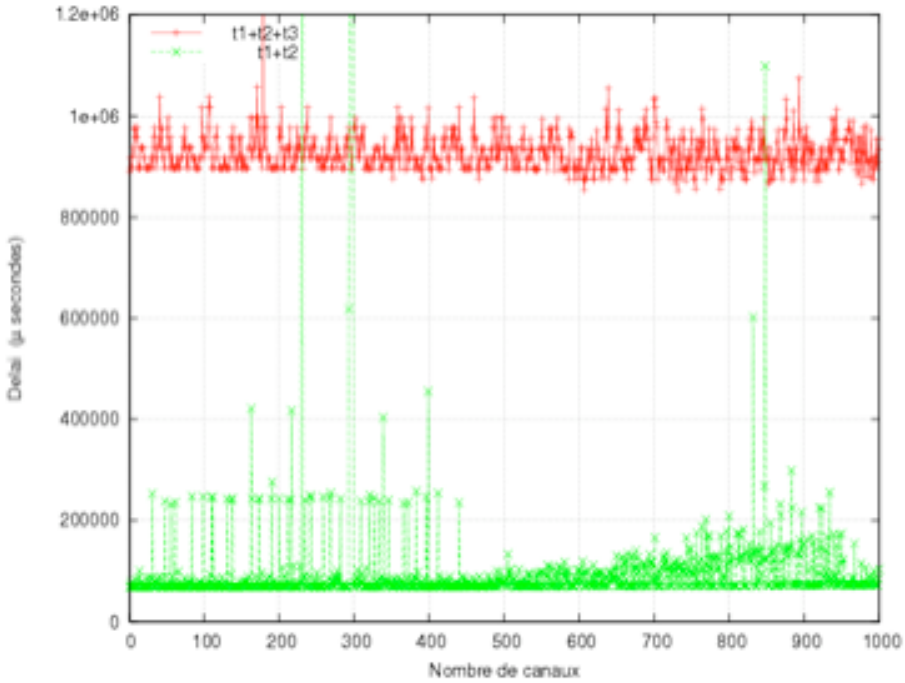



Figure 5. Delay variation source: Strasbourg, controller: Trondheim, receiver: Munich

Another question is whether we should use UDP (as we suggest) or TCP (as suggested by Lehtonen in his draft [MULTISSM] otherwise rather similar to *ssmsdp*). We have implemented a TCP mechanism, and Figure 5 gives some results. One can see that receivers are informed a little latter

(about 150 ms) due to the RTT time needed to set up a TCP Connexion. The increase in the total delay is not very important since the PIM join time is an order of magnitude larger.

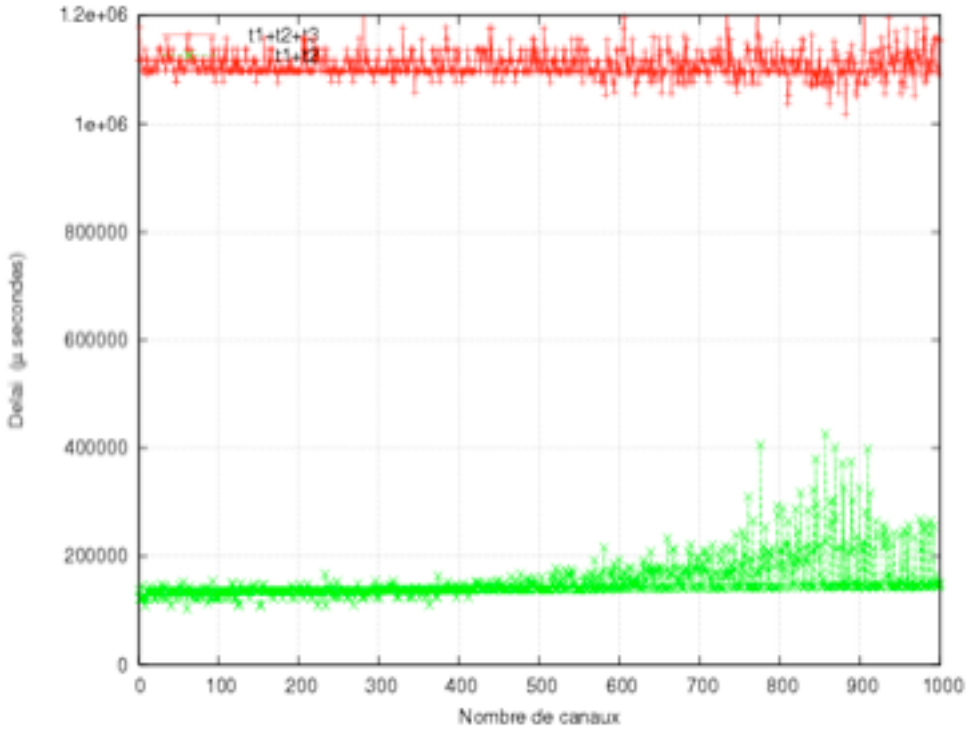



Figure 6. Delays when using TCP to send ON messages from source to controller

### 3.3.3. Redundant controllers

One potential problem with SSMSDP is that the controller of a session is a single point of failure. An architecture allowing several controllers and dynamically switching from a controller to another has been defined. The implementation is under way but it has not been tested yet. Another solution would be to use mechanisms such as *sas* (see section 2) where each source sends source announcements in a fully distributed way. This would need some further study on scalability issues.

## 3.4. The SSMSDPifier

Since SSMSDP gives almost the same service to the applications as ASM does, it is simple to port multiparty applications from ASM to SSM with SSMSDP. In fact a middleware called *ssmsdpifier* has been developed. It allows launching an ASM application from an *ssmsdpifier* process that will transparently translates ASM calls into SSMSDP and SSM calls. Note that since this tool uses *ssmsdp*, it can be used only on hosts with OS supporting MLDv2 (BSD or Linux currently). The *ssmsdpifier* has been successfully tested with applications such as *vic*, *NLANR beacon* and *dbeacon* (see section 5.2). As an example,

*prompt> ssmsdpifier –c 2001:660:4701::1 vic ff3e::8000:1234/1234*

launches vic with a controller 2001:660:4601::1, control channel ff3e::8000:1234 and port 1234, where vic is the classical tool designed to use ASM.

# 4. Congestion Control and reliability without feedback

Today most communications use a bidirectional channel even if data is unidirectional. The most prominent case is a TCP connection where acknowledgments are used for reliability (unacknowledged segments are retransmitted) and congestion control (a packet loss is interpreted as a possible congestion on the way, and the data rate is decreased by TCP). With multicast, a feedback channel is harder to obtain, at least for large groups. In particular there is the acknowledgment explosion problem if a large set of receivers send feedback to the source. This is why reliable multicast has long been an open problem. It seems that distinct solutions have to be found for reliability and for congestion control. Note also that with ASM every receiver can send feedback since an ASM communication is "bidirectional", whereas an SSM channel is unidirectional. If feedback must be sent, another channel or unicast must be used.

## 4.1. Reliability

For large groups reliability can be achieved without feedback. Error correction codes (FEC) can be used. In an N + K code, K redundant packets are transmitted in addition to the N initial packets. If at most K packets are lost, the N original packets can be reconstructed. Note that if more than K packets are lost, the original data cannot be recovered, but the receiver knows that data is missing. The Ietf has defined several building blocks to construct reliable multicast transmission. One building block is *flute* [FLUTE] which uses FEC and a unidirectional channel. Flute has been independently implemented at least twice : once by the Mad project in Finland [MAD-FLUTE] and once by Inria [INRIA-FLUTE]. Both have now been ported to IPv6 and to SSM. 6NET partners have already reported experiments. A tool named *mflute* has been developed to use *flute* for file mirroring with a periodic transmission of files to recover from failures, and a probabilistic unicast feedback to allow the source to have some indication of what is going on. From these preliminary experiments, one can conclude that *flute* is a good solution provided that we can solve the flow control problem.

## 4.2. Flow control and Congestion control

Flow control is an important feature since it allows a receiver to adapt the incoming rate to its capabilities or those of the network such that data can be received correctly. In unicast this means telling the source to reduce (or increase) its data rate. With multicast this is harder since the optimal rate might be different for each receiver. So the source should either pick a one-size fits all rate (at the risk that some receivers may not receive correctly), or send at several different rates. This last approach is used by the layered approach: data is sent at different speeds in different channels. For real time applications, such as video, higher layer brings more information (better images) when added to lower layers. In this case information carried by different layers are little redundant. A receiver adapts the received data rate (hence the video quality) by joining a number of layers (SSM channels for example). For non real time reliable applications such as file transfer, different layers are redundant, and the more layers one receives, the sooner one gets all data. In both cases, receivers change the data rate if packet losses are detected or not (using RTP for video for example). To adapt the number of received channels, the client must join or leave groups. Therefore these solutions depend on the delay imposed by IGMP/MLD and by PIM join/prune.

Congestion control is an even more important matter since it has an impact on other flows, including TCP flows. Therefore multicast transmissions must react quickly to congestion in order to not disrupt TCP flows (TCP friendliness).

Both *flute* implementations have a layering scheme to achieve congestion control. We have conducted some experiments on a local test bed using inria-flute and mad-flute in different settings to see how well they behave (fairness between flute sessions, fairness with TCP, convergence, influence of the layering mechanism).

### 4.3. Setup

The source of the tests is connected by a 100Mb/s link to a cisco 2600 router running PIM-SM v6 (routeur 2 in Figure 7). This router is connected by a 10 Mb/s (the bottleneck link) to a cisco 3825 router running PIM-SM and MLDv2 with explicit tracking (routeur 1 in Figure 7). The receiver is connected at 100 Mb/s to this last router. Explicit tracking is a feature where a router maintains the list of all receivers of a group. So when the last receiver leaves, the router may stop forwarding quickly, and send a PIM-prune upstream if necessary. Explicit tracking may be enabled or disabled depending on the experiments. When the receiver adds or removes a layer (SSM channel), it signals the router with MLDv2. We measure the data rate of all channels of all sessions in the receiver. We start 4 identical layered sessions at small intervals. A fifth session either identical to the first 4 or using TCP can be added later. The type of flow control ([RLC] or [FLID-SL]), the implementation (inria or mad) and the type of the fifth session are changed. Each flute session (and also the TCP connection) can use all the 10 Mb/s of the bottleneck link if launched alone.

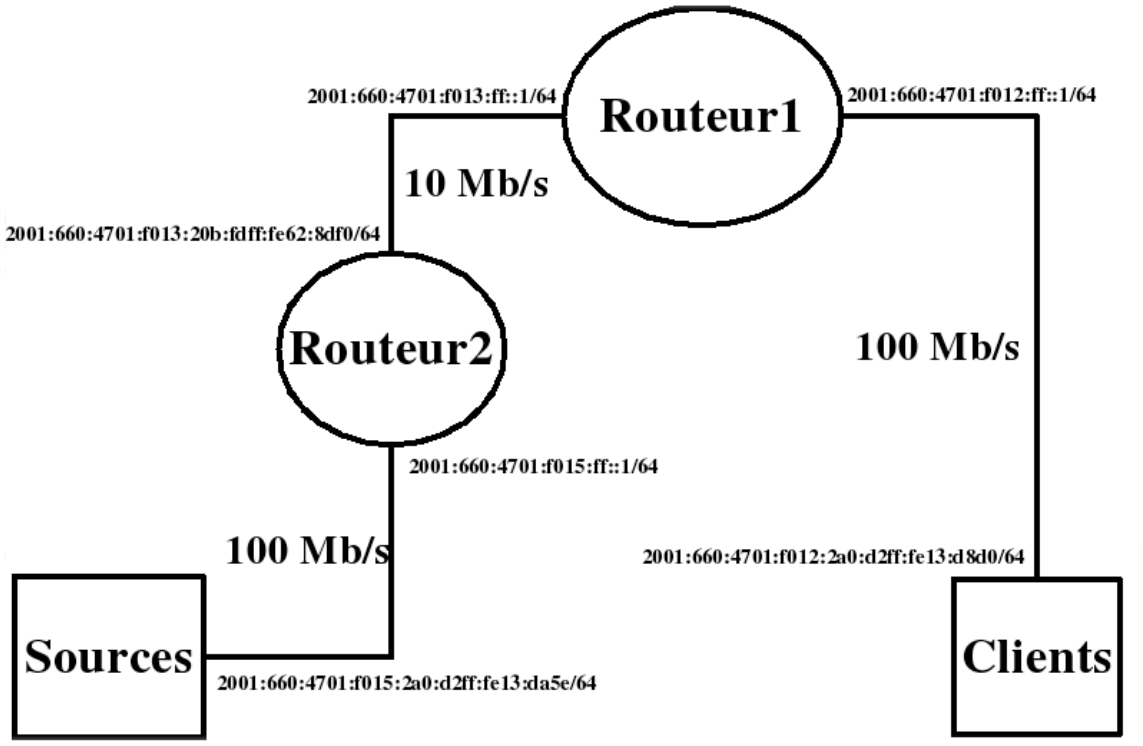



Figure 7. Setup for layered flute experiments

### 4.4. Results

The flute implementation from Mad currently implements only RLC (not FLID-SL), moreover it seems that there is a problem of convergence: when using parallel sessions, each session stays at a much lower rate than expected (the total rate is much less than the 10 Mb/s available). So we present results with the Inria implementation only.

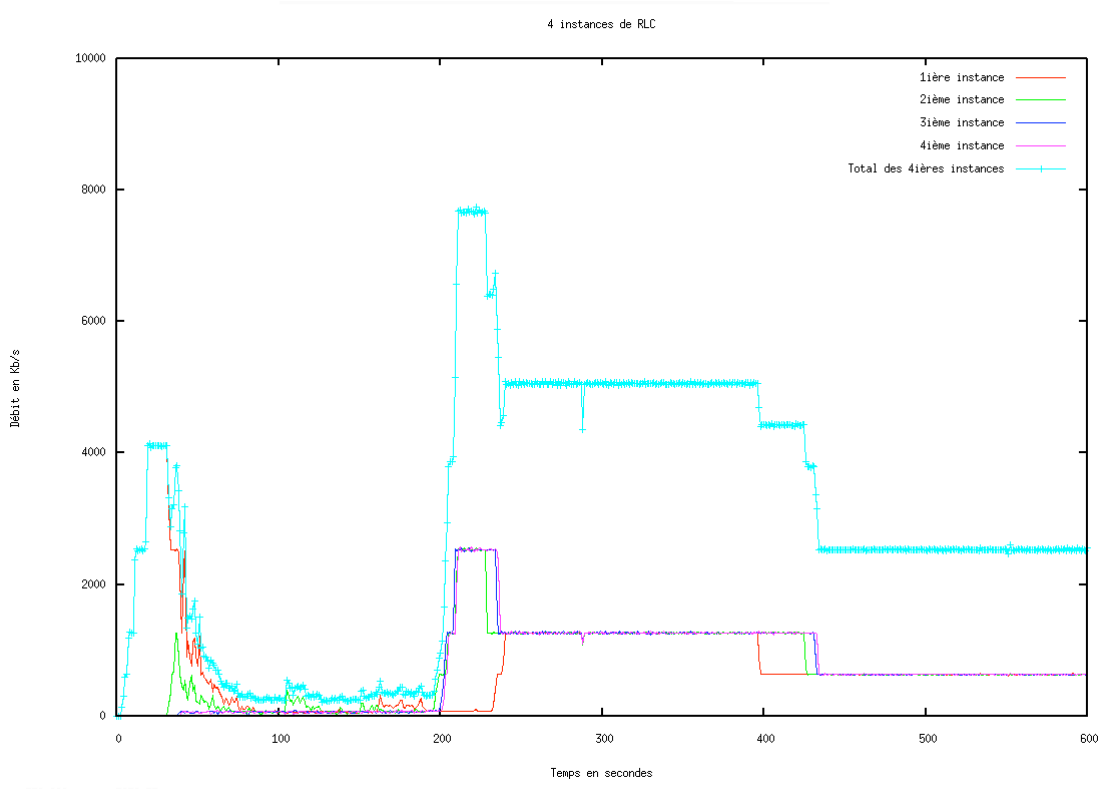



Figure 8 Four parallel sessions using RLC

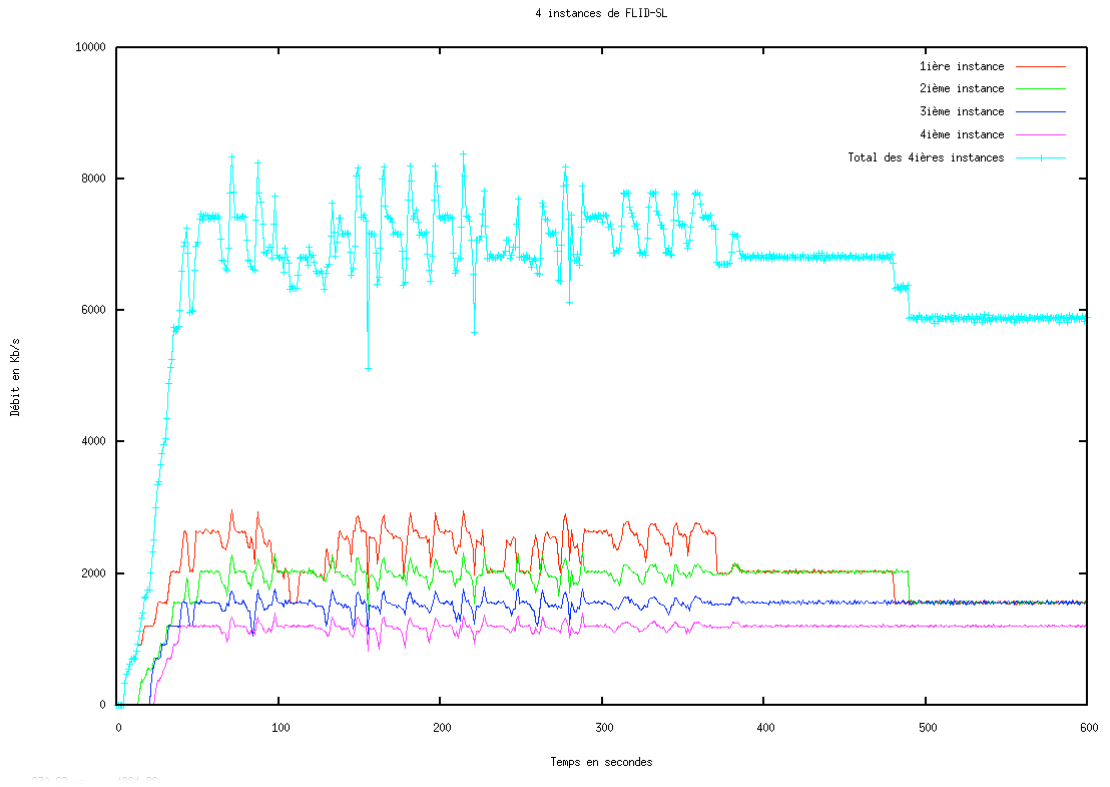



Figure 9 Four parallel sessions using FLID-SL

From Figure 8 and 9 it can be seen that FLID gives better results: the total bandwidth goes rapidly over 6 Mb/s, and bandwidth is fairly distributed among the 4 sessions (although sessions starting earlier keep an advantage). With RLC (Figure 8) total bandwidth stays very low for the first 200 seconds, and seems to peek at less than 5 Mb/s and then drops. The fairness between sessions is quite good. Note that developers of inria-flute informed us that their implementation of RLC is not fully compliant with the specifications, so this might explain these results.

In the next experiments, we start four sessions using RLC (Figure 10) or FLID (Figure 11) and then start a TCP session after 450 seconds (when multicast sessions should have stabilized)

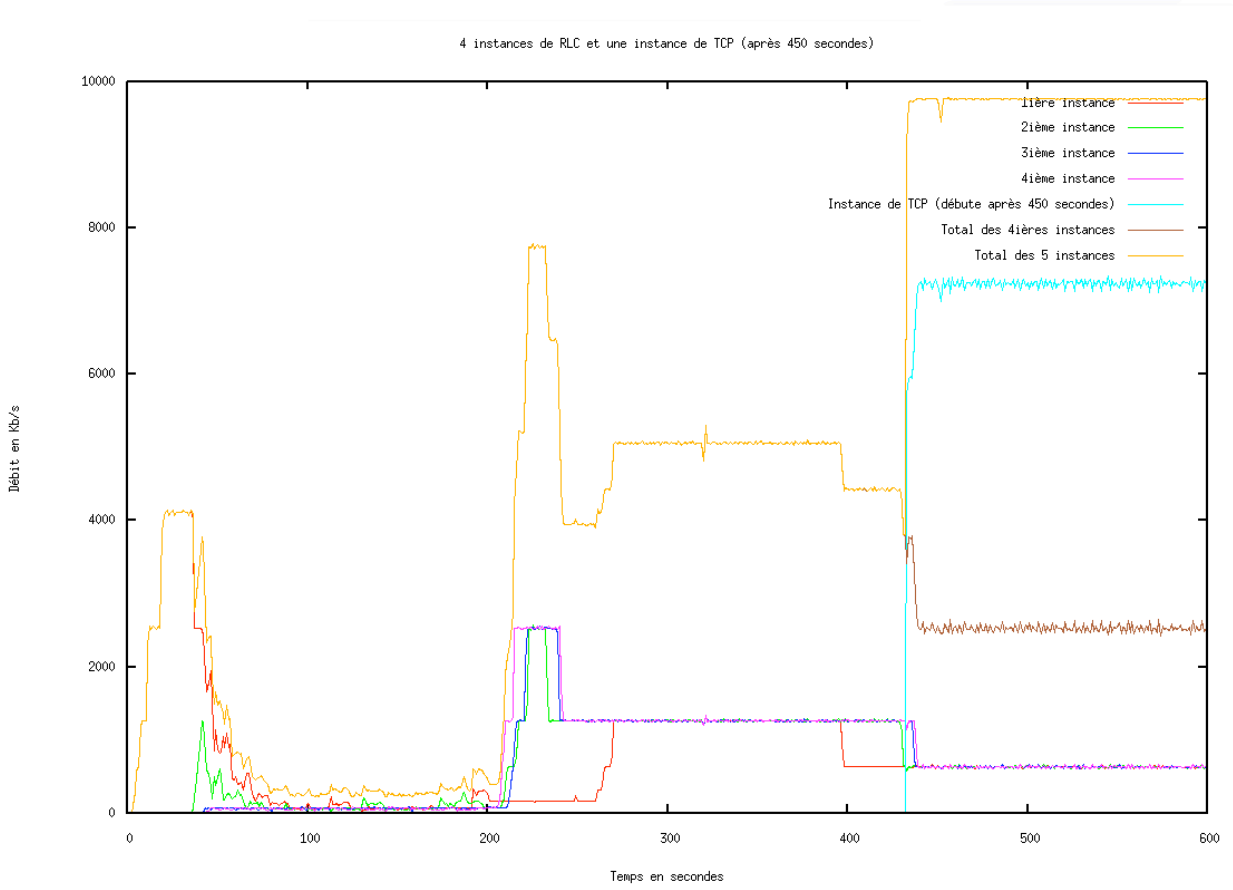



Figure 10 Four RLC multicast sessions with a TCP session starting after 450s.

It can be seen that with RLC, almost all available bandwidth is used, and TCP gets a big share: about 7.5 Mb/s where all 4 RLC sessions together get less than 2.5 Mb/s (RLC seems too TCP friendly). With FLID the bottleneck link is also fully used. However TCP gets a smaller share: about 3.5 MB/s and 6Mb/s for all multicast session. If we consider individual sessions, TCP has 3.5 Mb/s and a multicast session between 1 and 1.5 Mb/s, so TCP is winning. So FLID-SL is still TCP friendly. We did also some tests with or without explicit tracking, but preliminary results do not show significant differences. Additional tests are conducted to measure loss rates on the bottleneck link.

### 4.5. Conclusion on feedback

From these experiments it seems that some work must still be done to do better flow control. In any case if multicast is to be widely deployed, it is certainly better if there is no risk that it takes more than its share of bandwidth compared to TCP.

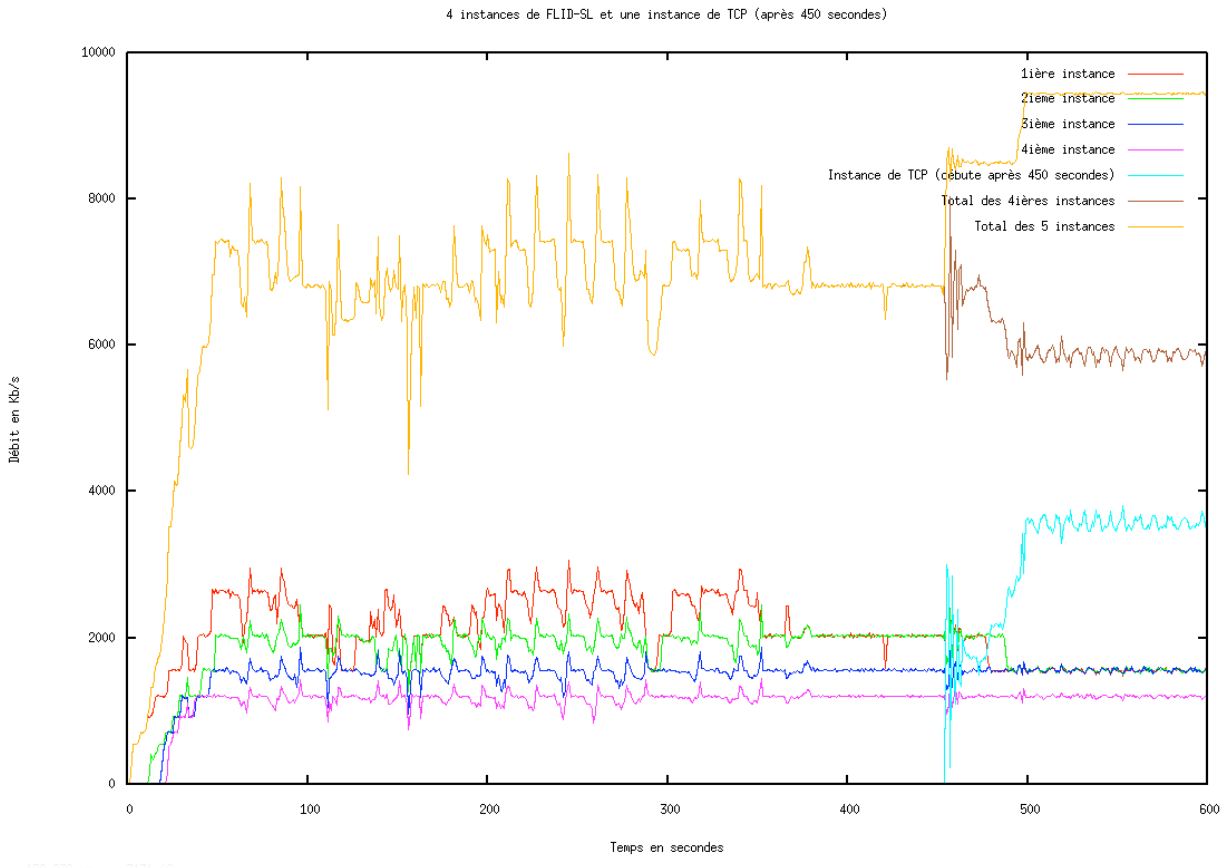



Figure 11 Four FLID multicast sessions with a TCP session starting after 450s

## 5. Testing Tools

One issue with deploying multicast applications is that multicast connectivity is not universally guaranteed. Therefore when a user launches a multicast application and nothing happens, it is quite hard to tell if:

- multicast connectivity is not available between sources ( or some sources) and receivers (or some receivers)
- there is a transient problem with multicast routing
- the source is not sending,
- or some other possibilities.

This issue is even harder with SSM since each channel in a multiparty application may have distinct problems. This is why tools are needed for users (at least advanced users) or local network managers to do some preliminary testing when an SSM session is not working properly.

### 5.1. ssmping

Prior to deploying SSM applications it might be useful to check that basic SSM connectivity is working. Also if there are problems with multicast connectivity, it can be useful to check whether one can receive SSM from other hosts. For testing unicast connectivity one often simply pings a host to verify that one can send to the host, and also receive from it. The tool *ssmping* [SSMPING] is an attempt at doing something similar for SSM. To ping a host one uses *ssmping*. The ping target must run *ssmpingd* in order to respond to queries. *ssmping* is run with a hostname or IP address as argument and gives output similar to *ping*. Based on the IP address S and a fixed group G, it will join the SSM channel (S,G) and send a unicast *ssmping* request to S. If S runs *ssmpingd*, this process will send both a unicast and a multicast reply back.

Example :

```
$ ssmping ssmping.uninett.no
ssmping joined (S,G) = (2001:700:1:7:211:d8ff:fe8f:1f9b,ff3e::4321:1234)
pinging S from 2001:630:d0:111:250:fcff:fe6a:42b3
  unicast from 2001:700:1:7:211:d8ff:fe8f:1f9b, seq=0 dist=20 time=57.106 ms
  unicast from 2001:700:1:7:211:d8ff:fe8f:1f9b, seq=1 dist=20 time=56.929 ms
  unicast from 2001:700:1:7:211:d8ff:fe8f:1f9b, seq=2 dist=20 time=62.466 ms
multicast from 2001:700:1:7:211:d8ff:fe8f:1f9b, seq=2 dist=12 time=65.706 ms
  unicast from 2001:700:1:7:211:d8ff:fe8f:1f9b, seq=3 dist=20 time=57.226 ms
multicast from 2001:700:1:7:211:d8ff:fe8f:1f9b, seq=3 dist=12 time=59.455 ms
  unicast from 2001:700:1:7:211:d8ff:fe8f:1f9b, seq=4 dist=20 time=56.090 ms
multicast from 2001:700:1:7:211:d8ff:fe8f:1f9b, seq=4 dist=12 time=58.956 ms

--- 2001:700:1:7:211:d8ff:fe8f:1f9b ssmping statistics ---
5 packets transmitted, time 4744 ms
unicast:
   5 packets received, 0% packet loss
   rtt min/avg/max/std-dev = 56.090/57.963/62.466/2.296 ms
multicast:
   3 packets received, 40% packet loss
      0% loss since first multicast packet received (after 2067 ms)
   rtt min/avg/max/std-dev = 58.956/61.372/65.706/3.077 ms
$
```

This shows that it took about 2s for the multicast tree to be established, since the first two multicast packets got lost. It also shows the round-trip time for the unicast and multicast (note that the

multicast is really unicast in one direction). This shows here that there is roughly 3-4ms additional delay for the multicast packets. It also shows that the unicast packets have travelled 20 hops, while multicast only 12 hops. This shows that there probably is some tunnelling for multicast. The hop distance is measured by having *ssmpingd* send all packets with a TTL/hop-limit of 64.

For further details, see [SSMPING].

### 5.2. Beacons

Another possibility is to have a tool to monitor SSM multicast routing by running beacons that periodically check multicast connectivity between them. The original NLANR multicast beacon uses an ASM groups. We have tested this beacon on top of SSMSDP to monitor SSM multicast connectivity. Similar tests have been made with the *dbeacon* [DBEACON]. Below are two snapshots, one with 24 participants using the regular *dbeacon* (Figure 12) with both ASM and SSM, and the second one with the *dbeacon* running on top of SSMSDP, with 5 participants (Figure 13).

As a conclusion on tools, we may say that with *ssmping* and/or *dbeacon* one can test if SSM connectivity is available between two hosts in the Internet (at least if a *beacon* or a *ssmpingd* daemon are running remotely). This should be sufficient for a user. However if it appears that connectivity is not available (or lost), these tools do not tell where the problem is. A tool similar to *traceroute* such as *mtrace* is missing for IPv6.

## 6. Conclusion

The SSM architecture is now well understood. PIM-SM for SSM and IPv6 is available on most routers as is MLDv2. On the client side, MLDv2 is still lacking on Windows OS, but should be available with the next (Longhorn) version. In the core of the network, SSM is much simpler to operate than ASM since it needs no RP and no additional protocol such as MSDP. Moreover these ASM mechanisms can be the target of DDoS attacks, whereas SSM seems more robust. By design there is no need for complex address allocations mechanisms since an SSM address must be unique for a given source only. Similarly it is easy to do source control since only one source may send in a channel.

For multiparty applications, there is a need to use either session relay where all sources send to an applicative relay that forwards data in an SSM channel, or to have some intra session source announcement. We have shown that with tools such as *ssmsdp*, it is possible to keep the same source discovery service at the application level, while using SSM only in the network. Moreover this can be done with a very minor modification to existing ASM applications. New functionalities could easily be introduced in the controller (floor control). One potential problem with this proposition is scalability in terms of sessions or sources. We have seen that this is not a problem, at least for sessions with a few hundred sources. One problem sometimes mentioned is the number of state that has to be maintained in routers if each source has its own tree. However this is not very different from ASM, since a source tree is usually constructed for each active source.

Another potential problem with SSM only is service discovery (or session announcement) using multicast, such as session announcements with sdr/SAP since this makes use of an ASM group that has no clear “owner” (hence no controller). Propositions such as *sas* can be used in this case. This should probably be extended to a hierarchical version for scalability reasons. In any case different tools are probably needed for different types of applications (TV channels, network games, videoconferencing).

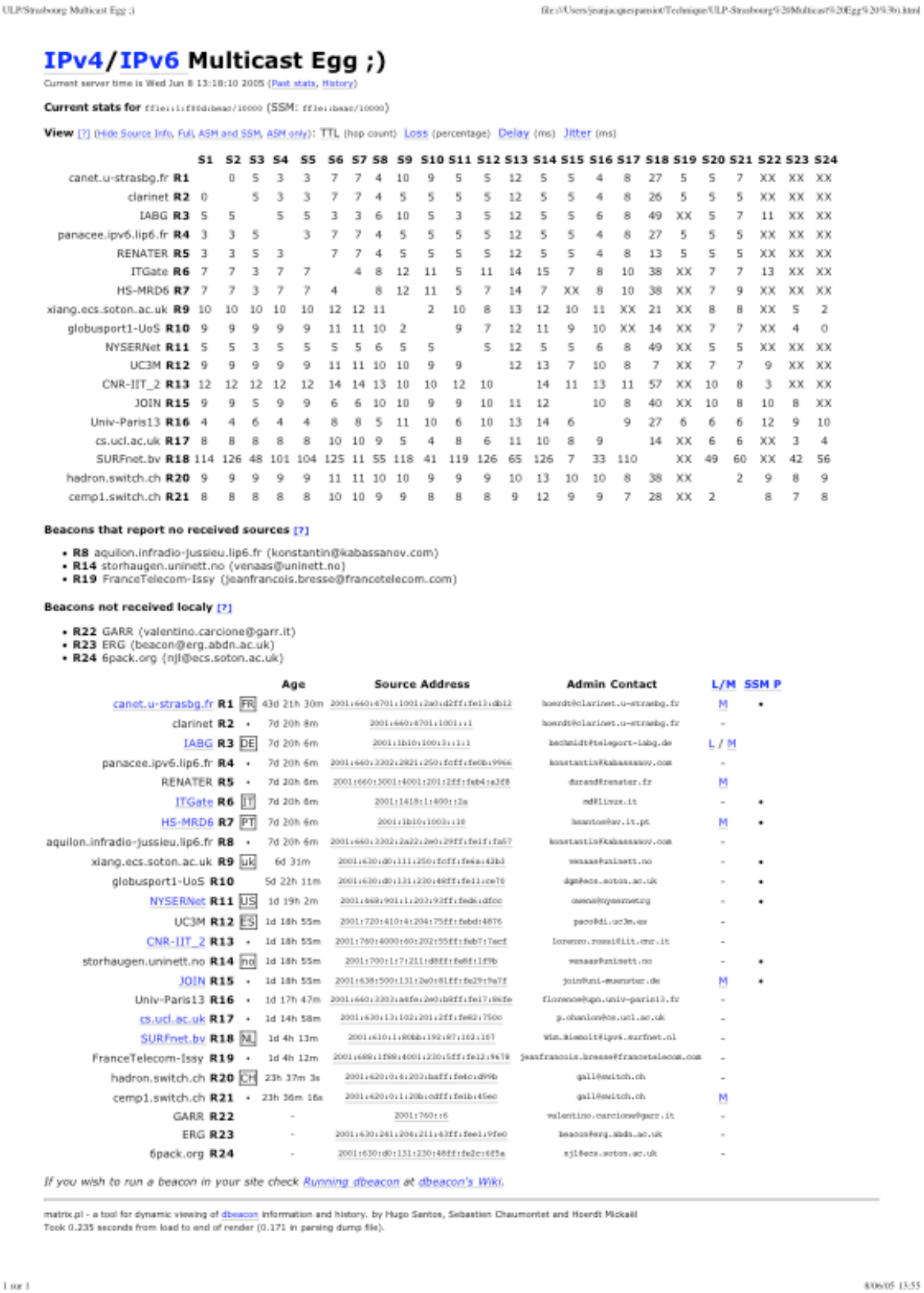
On the host side, MLDv2 is not yet widely available (Linux or BSD with Kame patch), it is still missing for Windows and MacOS, but should be available in the Longhorn release of Windows. As

far as applications are concerned, we have seen that ASM applications can be easily ported to SSM: relatively straightforward for single source applications, quite easy for multiparty applications when using *ssmsdp*. Reliable multicast can make use of a unidirectional channel when using FEC, as with flute.

Concerning flow and congestion control, solutions are available with layering and mechanisms such as FLID. They can be used both for video transmission and for reliable transfer. Some improvements are probably needed to guarantee a faster convergence and an improved fairness between flows. However a good TCP friendliness seems to be achieved, at least for connections with small RTTs. Eventually, all SSM applications (except very low bandwidth ones) should have this type of congestion control.

On the other hand there is still a lack in management tools for multicast in general and particularly for SSM over IPv6 in an inter-domain context. Tools such as *ssmping* or beacons are a first answer to check connectivity but they are not sufficient to locate connectivity problems precisely. Tools such as mtrace6 (*mtrace* is available for IPv4) or *tracetree* would be useful. Also Mibs allowing to remotely monitor an IPv6 multicast network are needed.

In conclusion we can say that SSM over IPv6 seems a very attractive solution for multicast, particularly at the inter-domain level. Almost all building blocks are there, waiting for some popular applications.



ULP/Strasbourg Multicast Egg ;)

# IPv4/IPv6 Multicast Egg ;)

Current server time is Wed Jun 8 13:18:10 2005 (Past stats, History)

**Current stats for** ff3e::1:f86d:beac/10000 (SSM: ff3e::beac/10000)

**View** [?] (Hide Source Info, Full, ASM and SSM, ASM only): TTL (hop count) Loss (percentage) Delay (ms) Jitter (ms)

| | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 | S11 | S12 | S13 | S14 | S15 | S16 | S17 | S18 | S19 | S20 | S21 | S22 | S23 | S24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| canet.u-strasbg.fr **R1** | | 0 | 5 | 3 | 3 | 7 | 7 | 4 | 10 | 9 | 5 | 5 | 12 | 5 | 5 | 4 | 8 | 27 | 5 | 5 | 7 | XX | XX | XX |
| clarinet **R2** | 0 | | 5 | 3 | 3 | 7 | 7 | 4 | 5 | 5 | 5 | 5 | 12 | 5 | 5 | 4 | 8 | 26 | 5 | 5 | 5 | XX | XX | XX |
| IABG **R3** | 5 | 5 | | 5 | 5 | 3 | 3 | 6 | 10 | 5 | 3 | 5 | 12 | 5 | 5 | 6 | 8 | 49 | XX | 5 | 7 | 11 | XX | XX |
| panacee.ipv6.lip6.fr **R4** | 3 | 3 | 5 | | 3 | 7 | 7 | 4 | 5 | 5 | 5 | 5 | 12 | 5 | 5 | 4 | 8 | 27 | 5 | 5 | 5 | XX | XX | XX |
| RENATER **R5** | 3 | 3 | 5 | 3 | | 7 | 7 | 4 | 5 | 5 | 5 | 5 | 12 | 5 | 5 | 4 | 8 | 13 | 5 | 5 | 5 | XX | XX | XX |
| ITGate **R6** | 7 | 7 | 3 | 7 | 7 | | 4 | 8 | 12 | 11 | 5 | 11 | 14 | 15 | 7 | 8 | 10 | 38 | XX | 7 | 7 | 13 | XX | XX |
| HS-MRD6 **R7** | 7 | 7 | 3 | 7 | 7 | 4 | | 8 | 12 | 11 | 5 | 7 | 14 | 7 | XX | 8 | 10 | 38 | XX | 7 | 9 | XX | XX | XX |
| xiang.ecs.soton.ac.uk **R9** | 10 | 10 | 10 | 10 | 10 | 12 | 12 | 11 | | 2 | 10 | 8 | 13 | 12 | 10 | 11 | XX | 21 | XX | 8 | 8 | XX | 5 | 2 |
| globusport1-UoS **R10** | 9 | 9 | 9 | 9 | 9 | 11 | 11 | 10 | 2 | | 9 | 7 | 12 | 11 | 9 | 10 | XX | 14 | XX | 7 | 7 | XX | 4 | 0 |
| NYSERNet **R11** | 5 | 5 | 3 | 5 | 5 | 5 | 5 | 6 | 5 | 5 | | 5 | 12 | 5 | 5 | 6 | 8 | 49 | XX | 5 | 5 | XX | XX | XX |
| UC3M **R12** | 9 | 9 | 9 | 9 | 9 | 11 | 11 | 10 | 10 | 9 | 9 | | 12 | 13 | 7 | 10 | 8 | 7 | XX | 7 | 7 | 9 | XX | XX |
| CNR-IIT_2 **R13** | 12 | 12 | 12 | 12 | 12 | 14 | 14 | 13 | 10 | 10 | 12 | 10 | | 14 | 11 | 13 | 11 | 57 | XX | 10 | 8 | 3 | XX | XX |
| JOIN **R15** | 9 | 9 | 5 | 9 | 9 | 6 | 6 | 10 | 10 | 9 | 9 | 10 | 11 | 12 | | 10 | 8 | 40 | XX | 10 | 8 | 10 | 8 | XX |
| Univ-Paris13 **R16** | 4 | 4 | 6 | 4 | 4 | 8 | 8 | 5 | 11 | 10 | 6 | 10 | 13 | 14 | 6 | | 9 | 27 | 6 | 6 | 6 | 12 | 9 | 10 |
| cs.ucl.ac.uk **R17** | 8 | 8 | 8 | 8 | 8 | 10 | 10 | 9 | 5 | 4 | 8 | 6 | 11 | 10 | 8 | 9 | | 14 | XX | 6 | 6 | XX | 3 | 4 |
| SURFnet.bv **R18** | 114 | 126 | 48 | 101 | 104 | 125 | 11 | 55 | 118 | 41 | 119 | 126 | 65 | 126 | 7 | 33 | 110 | | XX | 49 | 60 | XX | 42 | 56 |
| hadron.switch.ch **R20** | 9 | 9 | 9 | 9 | 9 | 11 | 11 | 10 | 10 | 9 | 9 | 9 | 10 | 13 | 10 | 10 | 8 | 38 | XX | | 2 | 9 | 8 | 9 |
| cemp1.switch.ch **R21** | 8 | 8 | 8 | 8 | 8 | 10 | 10 | 9 | 9 | 8 | 8 | 8 | 9 | 12 | 9 | 9 | 7 | 28 | XX | 2 | | 8 | 7 | 8 |

**Beacons that report no received sources** [?]

- **R8** aquilon.infradio-jussieu.lip6.fr (konstantin@kabassanov.com)
- **R14** storhaugen.uninett.no (venaas@uninett.no)
- **R19** FranceTelecom-Issy (jeanfrancois.bresse@francetelecom.com)

**Beacons not received localy** [?]

- **R22** GARR (valentino.carcione@garr.it)
- **R23** ERG (beacon@erg.abdn.ac.uk)
- **R24** 6pack.org (njl@ecs.soton.ac.uk)

| | | | Age | Source Address | Admin Contact | L/M | SSM | P |
|---|---|---|---|---|---|---|---|---|
| canet.u-strasbg.fr **R1** | FR | | 43d 21h 30m | 2001:660:4701:1001:2a0:d2ff:fe13:dbl2 | hoerdt@clarinet.u-strasbg.fr | M | • | |
| clarinet **R2** | • | | 7d 20h 8m | 2001:660:4701:1001::1 | hoerdt@clarinet.u-strasbg.fr | - | | |
| IABG **R3** | DE | | 7d 20h 6m | 2001:1b10:100:3::1:1 | bschmidt@teleport-iabg.de | L / M | | |
| panacee.ipv6.lip6.fr **R4** | • | | 7d 20h 6m | 2001:660:3302:2021:250:fcff:fe0b:9966 | konstantin@kabassanov.com | - | | |
| RENATER **R5** | • | | 7d 20h 6m | 2001:660:3001:4001:201:2ff:feb4:a3d8 | durand@renater.fr | M | | |
| ITGate **R6** | IT | | 7d 20h 6m | 2001:1418:1:400::2a | md@linux.it | - | • | |
| HS-MRD6 **R7** | PT | | 7d 20h 6m | 2001:1b10:1003::10 | kuanton@av.it.pt | M | • | |
| aquilon.infradio-jussieu.lip6.fr **R8** | • | | 7d 20h 6m | 2001:660:3302:2a22:2e0:29ff:fe1f:fa57 | konstantin@kabassanov.com | - | | |
| xiang.ecs.soton.ac.uk **R9** | uk | | 6d 31m | 2001:630:d0:111:250:fcff:fe6a:42b3 | venaas@uninett.no | - | • | |
| globusport1-UoS **R10** | | | 5d 22h 11m | 2001:630:d0:131:230:48ff:fe11:ce76 | dgm@ecs.soton.ac.uk | - | • | |
| NYSERNet **R11** | US | | 1d 19h 2m | 2001:468:901:1:203:93ff:fed6:dfcc | owens@nysernet.org | - | • | |
| UC3M **R12** | ES | | 1d 18h 55m | 2001:720:410:4:204:75ff:febd:4876 | paco@di.uc3m.es | - | | |
| CNR-IIT_2 **R13** | • | | 1d 18h 55m | 2001:760:4000:60:202:55ff:feb7:7acf | lorenzo.rossi@iit.cnr.it | - | | |
| storhaugen.uninett.no **R14** | no | | 1d 18h 55m | 2001:700:1:7:211:d8ff:fe8f:1f9b | venaas@uninett.no | - | • | |
| JOIN **R15** | • | | 1d 18h 55m | 2001:638:500:131:2e0:81ff:fe29:9a7f | join@uni-muenster.de | M | • | |
| Univ-Paris13 **R16** | • | | 1d 17h 47m | 2001:660:3303:a6fe:2e0:b8ff:fe17:86fe | florence@lpn.univ-paris13.fr | - | | |
| cs.ucl.ac.uk **R17** | • | | 1d 14h 58m | 2001:630:13:102:201:2ff:fe62:750c | p.ohanlon@cs.ucl.ac.uk | - | | |
| SURFnet.bv **R18** | NL | | 1d 4h 13m | 2001:610:1:80bb:192:87:162:167 | Wim.Biemolt@ipv6.surfnet.nl | - | | |
| FranceTelecom-Issy **R19** | • | | 1d 4h 12m | 2001:688:1588:4001:230:5ff:fe13:9678 | jeanfrancois.bresse@francetelecom.com | - | | |
| hadron.switch.ch **R20** | CH | | 23h 37m 3s | 2001:620:0:4:203:baff:fe4c:d99b | gall@switch.ch | - | | |
| cemp1.switch.ch **R21** | • | | 23h 36m 16s | 2001:620:0:1:20b:cdff:fe1b:45ec | gall@switch.ch | M | | |
| GARR **R22** | | | - | 2001:760:16 | valentino.carcione@garr.it | - | | |
| ERG **R23** | | | - | 2001:630:241:206:211:43ff:fee1:9fe0 | beacon@erg.abdn.ac.uk | - | | |
| 6pack.org **R24** | | | - | 2001:630:d0:131:230:48ff:fe2c:6f5a | njl@ecs.soton.ac.uk | - | | |

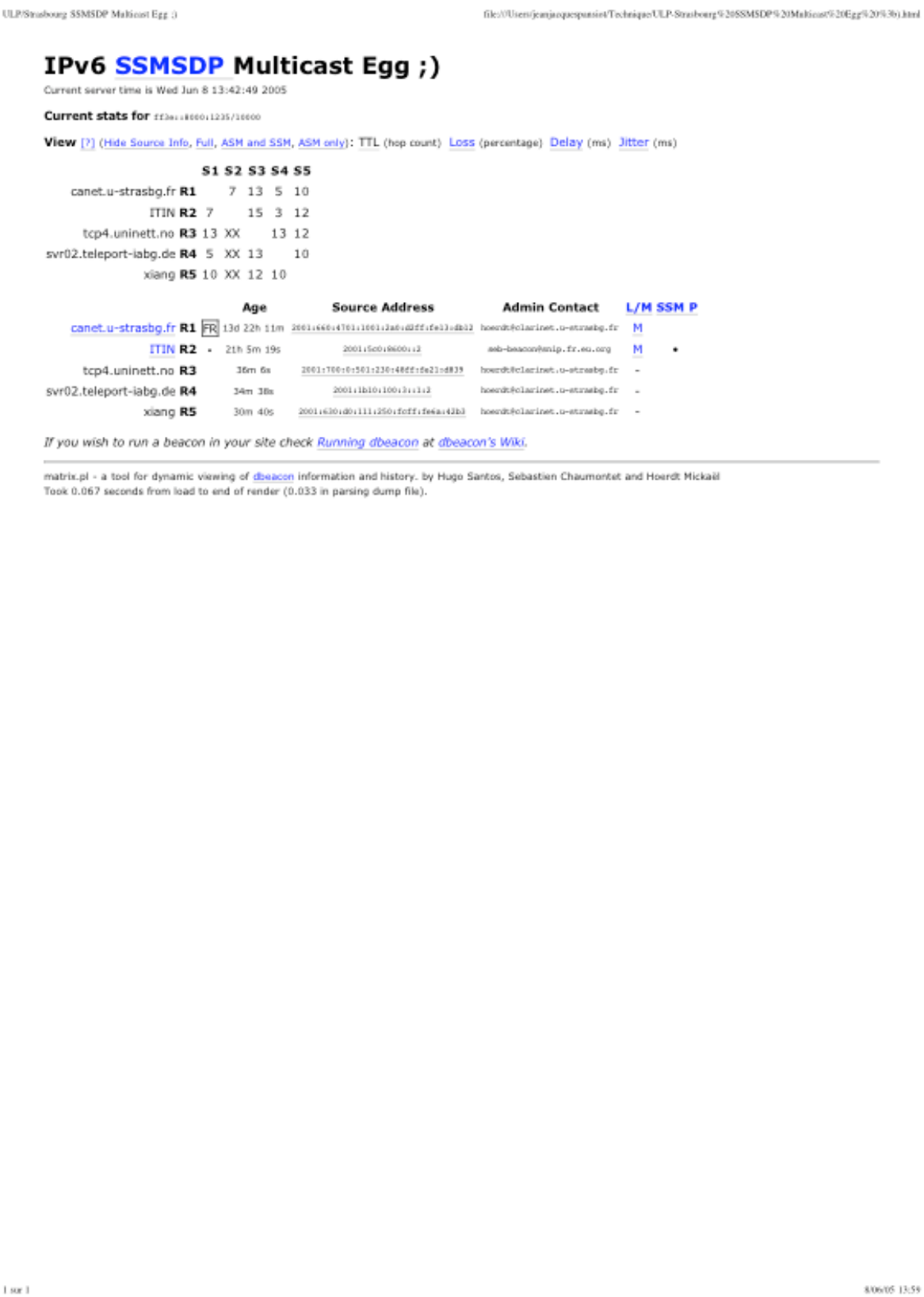
*If you wish to run a beacon in your site check Running dbeacon at dbeacon's Wiki.*

matrix.pl - a tool for dynamic viewing of dbeacon information and history. by Hugo Santos, Sebastien Chaumontet and Hoerdt Mickaël
Took 0.235 seconds from load to end of render (0.171 in parsing dump file).

1 sur 1

8/06/05 13:55

Figure 12 Multicast Connectivity matrix with dbeacon



ULP/Strasbourg SSMSDP Multicast Egg ;) file:///Users/jeanjacquesputaist/Technique/ULP-Strasbourg%20SSMSDP%20Multicast%20Egg%20(%3b).html

# IPv6 SSMSDP Multicast Egg ;)

Current server time is Wed Jun 8 13:42:49 2005

**Current stats for** ff3e::8000:1235/16000

**View** [?] (Hide Source Info, Full, ASM and SSM, ASM only): TTL (hop count) Loss (percentage) Delay (ms) Jitter (ms)

| | | S1 | S2 | S3 | S4 | S5 |
|---|---|---|---|---|---|---|
| canet.u-strasbg.fr | R1 | | 7 | 13 | 5 | 10 |
| ITIN | R2 | 7 | | 15 | 3 | 12 |
| tcp4.uninett.no | R3 | 13 | XX | | 13 | 12 |
| svr02.teleport-iabg.de | R4 | 5 | XX | 13 | | 10 |
| xiang | R5 | 10 | XX | 12 | 10 | |

| | | | Age | Source Address | Admin Contact | L/M | SSM | P |
|---|---|---|---|---|---|---|---|---|
| canet.u-strasbg.fr | R1 | FR | 13d 22h 11m | 2001:660:4701:1001:2a0:d2ff:fe13:db12 | hoerdt@clarinet.u-strasbg.fr | M | | |
| ITIN | R2 | - | 21h 5m 19s | 2001:5c0:8600:12 | aeb-beacon@snip.fr.eu.org | M | * | |
| tcp4.uninett.no | R3 | | 36m 6s | 2001:700:0:501:230:48ff:fe21:d839 | hoerdt@clarinet.u-strasbg.fr | - | | |
| svr02.teleport-iabg.de | R4 | | 34m 38s | 2001:1b10:100:3::1:2 | hoerdt@clarinet.u-strasbg.fr | - | | |
| xiang | R5 | | 30m 40s | 2001:630:d0:111:250:fcff:fe6a:42b3 | hoerdt@clarinet.u-strasbg.fr | - | | |

*If you wish to run a beacon in your site check Running dbeacon at dbeacon's Wiki.*

matrix.pl - a tool for dynamic viewing of dbeacon information and history. by Hugo Santos, Sebastien Chaumontet and Hoerdt Mickaël
Took 0.067 seconds from load to end of render (0.033 in parsing dump file).

1 sur 1 8/06/05 13:59

Figure 13 SSM connectivity matrix with dbeacons running ssmsdp

# 7. References

[DBEACON] http://artemis.av.it.pt/~hsantos/dbeacon/

[FLID] FLID-DL: Congestion Control for Layered Multicast, John Byers, Michael Frumin, Gavin Horn, Michael Luby, Michael Mitzenmacher, Alex Roetter, and William Shaver, in Second Int'l Workshop on Networked Group Communication (NGC 2000).

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

[GLOBECOM05] Highly Dynamic SSM Source Discovery Protocol, Mickaël Hoerdt and Jean-Jacques Pansiot, Globecom 2005.

[ICC04] Multicast Session Announcements on top of SSM, P. Namburi and K. Sarac, http://www.utdallas.edu/~ksarac/research/publications/ICC04.pdf

[INRIA-FLUTE] http://www.inrialpes.fr/planete/people/roca/mcl/flute_infos.html

[MAD-FLUTE] http://www.atm.tut.fi/mad/

[MULTISSM] R. Lehtonen, Dynamic Multi-Source Discovery for SSM using MSDP, October 2004. draft-lehtonen-mboned-multissm-01.txt

[RLC] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like Congestion Control for Layered Multicast Data Transfer. In Proc. IEEE INFOCOM, pages 996-1003, San Francisco, CA, March 1998.

[SAP] session announcement protocol RFC 2974

[SASD] http://www.venaas.no/multicast/sas/

[SSM-DISC] Requirements for discovery of dynamic SSM sources, draft-lehtonen-mboned-dynssm-req-00.txt, R. Lehtonen, TeliaSonera, S. Venaas, University of Southampton, M. Hoerdt, University Louis Pasteur – LSIIT, Feb 11, 2005.

[SSMPING] http://www.venaas.no/multicast/ssmping/

[SSMSDP] http://clarinet.u-strasbg.fr/~hoerdt/libssmsdp/, see also Source Discovery Protocol in SSM Networks (draft-beck-mboned-ssm-source-discovery-protocol-03.txt), Frédérick Beck, Mickaël Hoerdt, Jean-Jacques Pansiot.