Access control in multicast environments: an approach to senders authentication

Antonio F. Gómez Skarmeta Angel L. Mateo Martínez
Pedro M. Ruíz Martínez

Computer Science, Electronic and Artificial Intelligence Dept.
University of Murcia
Campus de Espinardo, s/n 30008 Murcia

{skarmeta,amateo}@dif.um.es
pedro.ruiz@rediris.es [*]

2nd November 1999

Abstract

During the last years, a lot of new interactive multimedia tools based on IP multicast over the MBone have been developed. Although MBone makes possible audio and video transmission over the Internet, it is used only by people involved in the ongoing deployment and management of the IP multicast technology. MBone has a lot of problems like wasted bandwidth, no control over who is joined to a session, there is no way to control the scope of sent packets, etc. This kind of problems are making Internet Service Providers (ISP) to think twice before offering multicast to their clients. So we are trying to solve this kind of problems, especially those related with multicast senders control and authentication, by modifying multicast routing schemes.

1 Introduction

Internet has undergone lots of improvements during the last years, making possible the development of new interactive services like videoconferencing. However, these services are bandwidth intensive and could clog our network, so a good network topology is essential to get a proper resource usage.

Recent popular applications, like *NetMeeting* or *CU-SeeMe*, are based on the concept of "reflector" when dealing with multipart conferences. A reflector provides for the replication of streams, that is to say, after receiving a datagram from a participant, it sends a copy of that datagram to the rest of participants in the videoconference.

As an alternative to the use of this kind of applications, there is another family of tools known as MBone tools. All these tools use IP multicast[2] instead of traditional IP (unicast IP). IP multicast

[*]Pedro M. Ruiz was a researcher of the University of Murcia during the development of this work. At this moment, he is working at RedIRIS, the Spanish National Research Network

technology uses group addresses instead of host addresses belonging to a unique host. So, in order to be IP multicast compliant, an Internet host must be able to receive packets addressed to its IP unicast address and packets addressed to some of the groups it is a member. With this new model, new routing equipment and new routing protocols are needed to manage this traffic. Using this service model we can avoid the use of reflectors and we can reduce the bandwidth used by multimedia communications.

These new multicast routing equipments, usually called *mrouters*, act like normal routers. The main difference is that they use a multicast routing algorithm besides a unicast routing protocols. When such a router receives a packet addressed to a multicast group, it selects the output interfaces that must be used to forward the packet according to the information stored in its internal multicast routing table. We can distinguish between two kinds of *mrouters*: normal routers extended with multicast capabilities and workstations running a multicast routing daemon (MRD).

Nowadays, not every router in the Internet is a multicast-capable router. Between while, we need to use tunnels to encapsulate multicast packets over non multicast capable networks. In these situations, multicast packets are encapsulated into a unicast packet addressed to the *mrouter* in the other end of the tunnel. All this virtual multicast network connected via these tunnels and multicast-capable router is known as *Multicast Backbone* (*MBone*[1]).

During the last years, there has been a great effort to develop new streaming media applications, a suite of new multicast routing algorithms or even new real-time protocols. However, there are some other issues that, although they are less attractive, they are very important for *MBone* to become a totally settled technology and a suitable service for ISPs. Some of this issues are: access control over members of certain multicast group, control over scope or *Time To Live* (TTL) that a sender can use send packets and control over who is allowed to transmit to certain multicast group.

Taking these problems into account, IP multicast seems to be a complex, difficult to manage and difficult to use service. So, service providers or even universities prefer not to offer it to their customers or students, and then *MBone* is not having the importance it should have.

Nowadays, routing algorithms have a high level of reliability. Once you decide the network topology to use and you configure the multicast-capable router the system use work fine. When problems appears it is very tedious to debug because of the lack of multicast management tools.

However, although network management is not difficult once it is correctly configured, ISPs remain without offering IP multicast as a product. The main reason is that there is no way to control who access to MBone and how he uses it. So, service providers can't offer it like an additional service because they have only two options: They can offer it to everybody or they can offer it to nobody. In the same way, universities offering the MBone to all of their students could get their critical traffic disrupted.

In order to achieve a widespread use of the IP multicast technology with all the benefits and advantages that it offers, there must be defined some mechanism to manage who uses it and how uses it. Using this mechanism, providers could offer the service to their customers in the same way they offer e-mail or WWW these days.

The main problem we try to solve is access control to IP multicast environments. Our system should provide control over who connects to MBone, who sends data to what multicast group, who receives certain multicast group. However, this would be a definitive solution that takes time to implement. So we have developed an intermediate solution allowing us to authenticate multicast senders, but it has not considerations about receivers.

The paper proceeds as follows: Section 2 discusses the general idea behind a multicast router and its relationship with the kernel of the Operating System (OS). Section 3 describes the different solutions that we have been developing. It also explains the different decisions we took during the development and makes a much more detailed description of the final solution that we offer. Section 4 presents the conclusions of the work and describes some future work that we are planning in order to improve the final solution.

2 The work of the MRD

Although there are a lot of multicast routing algorithms, like PIM and DVMRP [3], all of them work in a similar way. They must select the virtual interfaces (VIF) that need be used to forward the multicast packets that the *mrouter* receives.

To improve scalability and slow down the bandwidth used, some of these algorithms use a mechanism called *pruning*. Firstly, *pruning* tries to forward multicast packets using only the VIFs that have interested receivers. But, how does MRD know if there are hosts in the network attached to that VIF being interested in certain multicast group?. The Internet Group Management Protocol (*IGMP* [4]) defines this interaction between hosts and *mrouters*. A host must use this protocol to tell the MRD that it wants to join to certain multicast group. And, mrouters have to register the multicast groups that have members on each VIF. As we will see the every MRD implementation is based on the forwarding support provided by the multicast-capable OS.

2.1 Relation between multicast routing and operating system

As we have said, multicast packets are frequently used in real-time and multimedia communications. This implies that the time used to process these packets may be minimized as much as possible. In order to reduce this time, MRD developers are allowed to ask the kernel for automatic forwarding of the packets. In this way, the MRD does not need to examine every packet.

Using this OS capability, the MRD must attend to IGMP messages to know what multicast packets need to be forwarded to what VIFs. Once it knows that, it uses a kernel system call, to ask the kernel for automatic forwarding of all packets received from that source address and addressed to that group using certain VIFs. Then OS will be the responsible of sending out these datagrams while MRD doesn't tell it to stop forwarding that datagrams. In the same way, when all the hosts attached to certain VIF of the *mrouter* leave the multicast group, MRD asks the OS, using another system call, to stop forwarding that packets.

The main problem here is that there is no control over multicast senders. Every user having an *mrouter* in his LAN can send multicast packets whenever he wants, using the TTL he wants and addressing the datagrams to the multicast group he wants. The key issue here is that the *mrouter* does not process these datagrams. It only ask the kernel to forward the packets according to the input VIF instead of using some kind of tag to identify the sender. So our main goal is to add such control to MRD.

This mode of forwarding multicast packets forces us to work in the kernel code because at the MRD level we can't control multicast datagrams. Thinking in the MRD code, we can only manage which multicast groups have to be forwarded, but we cannot decide which senders are allowed to send packets. There is no way to control users nor hosts. At this level there are two choices: we can let all the users to be joined to certain group or we can let nobody to join the group. This is not useful because we cannot distinguish between different users. In addition, if we let users to send datagrams to multicast groups, any host connected to the same *mrouter* by the same VIF, could send packets with any TTL it wants, and we can't do anything to limit that TTL. If we decide not to work in the kernel code, we get a solution being general enough to be implemented in every system using MRD.

The OS type doesn't matter in this case. However, this approach has a big drawback: it is only useful in very restricted environments.

So, we need a better solution based on the use of an open-source OS. Linux [5] or FreeBSD.

Nowadays, the management of multicast senders (MS) is a new and not developed issue. In fact, we have only found one proposal trying to solve the problem (at this moment as an Internet-draft [8]). This draft describes some extensions to IGMPv2 protocol to authenticate multicast senders and receivers.

These extensions allows a MRD (called ingress and egress routers) to authenticate MS via some external mechanism.

3 Developed solution

To solve these problems, we can offer a few solutions each of them having its advantages and disadvantages. But all possible solutions share the need to modify or improve the services that the OS offers to the MRD. As we have seen, this need is imposed by the way the OS forwards the packets. The basic idea behind this development is that we need to control when a certain packet has to be forwarded and when it hasn't. A multicast datagram has only to be forwarded when it meets a set of conditions that we have imposed on the user or host that sends the packet.

3.1 Kernel modifications to OS

The only way to control the multicast packets arriving to our *mrouter* is adding services to the OS kernel allowing MRD to ask for filtering of multicast packets.

These changes has been made in a PC with Linux RedHat 5.2 OS which acts as a *mrouter*. We started working on the 2.0.36 kernel release but we found some multicast-related problems in this version and we decided to upgrade to version 2.2.2.

We have added some data structures to the kernel. This structures are used to store the parameters related with the MS and also to keep a list of authorized multicast senders with the multicast groups they are authorized to send to. In the same way, we have added some functions to check if the traffic originated by a MS meets the authorized parameters for that users. These functions also allow us to discard packets not accomplishing such conditions. Furthermore, we have added new system calls allowing the MRD to dynamically reconfigure these filters.

In all the kernel versions we are using, we need to define some data structures allowing us to store the data related with the management of which users or hosts can send data, the multicast groups he can use and the TTL they can use. The data structure that we have created for this purpose is called *msender_ctl*. It stores information needed to control traffic generated from a multicast sender.

The next step was to really check that the traffic generated by our MS meets the parameters established for such MS in its corresponding *msender_ctl* structure. We started defining a control policy. The policy we have implemented in our system is the following: the *mrouter* drops all the packets coming from its internal network except those coming from a sender registered in kernel structures and meet the parameters associated with that sender.

Such a policy ensures that nobody can send packets addressed to a certain multicast group except those specific MS that we have enable. Therefore, users can't send data to unauthorized sessions and so, they can't maliciously interfere with existent multicast sessions.

Moreover, the system ensures that users are limited to an specified TTL (assigned by the administrator) because if they use a higher TTL, their packets will be dropped. With this scheme, a university could offer a free access to MBone or even MBone videoconferencing rooms without any risk. In the actual MBone, users can transmit with the TTL they want and using as much bandwidth as they want. So, they could easily clog our network or even networks outside our organization. In our newer scheme, a manager could assure i.e. that users will never send datagrams outside our network.

Once we have defined the policy, the next step is to implement it. The policy implementation is based on a function that we have added to the OS kernel that checks every packet in the forwarding queue of the OS. If a packet meets the previous policy, it is then allowed to be forwarded. In other case, it is silently dropped.

Although we have made some advances from the starting point, it is not enough. We must address another issue: How do we pass the parameter liked to a user to the kernel? Creating new system calls.

We have created three new system calls: first call is used to create the data structures for storing multicast senders' parameters, the second one is used to insert multicast a multicast sender from the list. These three functions provide us a huge flexibility because we can dynamically add and remove multicast senders from kernel structures without needing to restart *mrouter*.

3.2 The first solution

Once we have implemented the basic support to control multicast datagrams according to the defined policy, the next step is to mix all this parts in a real system and that the final system will work.

To be exact, this first solution consists in doing some additions of the source code of the MRD *mrouted*. We have added new code to allow it to read a configuration file called *mauth.conf* which was composed by a list of users and related parameters. We have modified version 3.9-beta3 of *mrouted* running over a Linux RedHat 5.2 with the version 2.2.2 of the kernel.

When this MRD is started, it reads data from this configuration file, using the added system calls to insert the authorized multicast senders into the kernel structures. After this, is starts routing multicast packets in the usual way. When it stopped, it previously removes these senders from the kernel tables. Therefore, to update the authorized sender list, you just have to send a HUP signal ("*kill -HUP*") to MRD after making the updates in the *mauth.conf* configuration file.

This has a clear benefit: it simplifies the management process. The only management issue is to edit a configuration file and then restart the daemon.

But this also has some drawbacks. To sum up , we would say that it doesn't use all the benefits provided by the system calls we have created.

3.3 An improved solution

The previous solution shows some drawbacks caused by the link we make between MRD and the process used to define parameters and users. In order to avoid these problems, we are going to move to a distributed approach. That's to say, we are going to unlink MS management from MRD operation mode. This way, we get a MRD independent architecture.

The system calls we have made, offers us some advantages allowing us to get this independence. These advantages are:

1. Our system calls, allow us to insert or delete MS without restarting anything. With the previous solution we had to restart MRD every time we made a new update on the parameters attached to a MS.

2. Our system calls, give us total independence from the MRD used. So, we don't need to make any changes to the routing algorithm's source code.

To achieve our goal, we are going to make our own user management daemon called *musersd* (Multicast users daemon).

Musersd is going to be continuously receiving commands addressed to an specified UDP port in its machine. Every time a new command arrives, *musersd* will execute it using our system calls.

Of course, the commands types are adding a new MS with its parameters or deleting a MS from the kernel. We have not added the creation of the structures as a command because the structure's startup process will be automatically made by *musersd* in its initial stage.

The management is made using a Java Applet that we have created. This applet is used to connect the *musersd* and the manager. We can see this applet like an interface between the system manager and the kernel of the machine that acts like a *mrouter*. So this applet, called *mauthclient*, is used by the manager to configure the parameters attached to each MS. These parameters will be updated in the *mrouter's* kernel.

Obviously, the first problem we must solve is the definition of a communication protocol between *musersd* and *mauthclient*. To solve the problem, we defined a *call/response* protocol with *musersd* acting as server and *mauthclient* acting as client. This communication scheme is shown in Fig. 1.

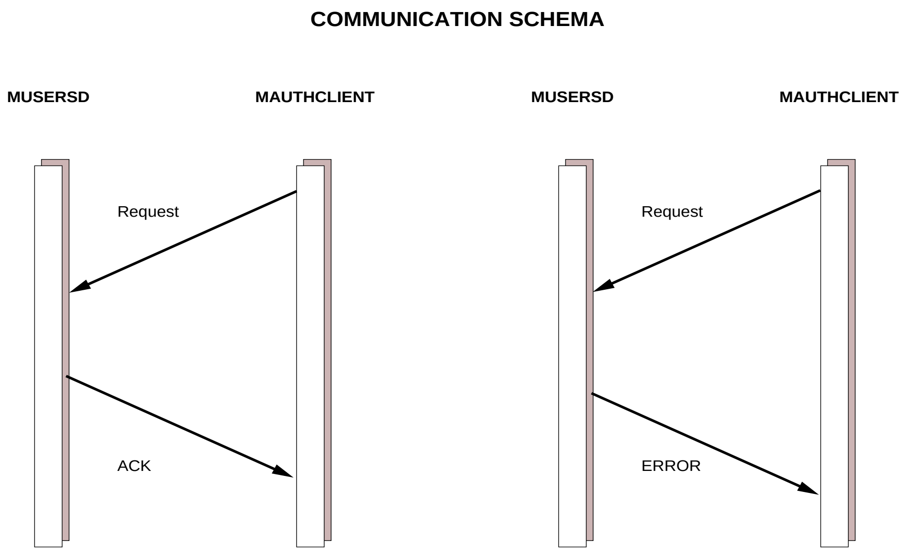

Figure 1: Communication protocol.

The client (*mauthclient*) will send the commands cited above and the server (*musersd*) will reply with an ACK or ERROR message depending on the result of the command.

In order to avoid side effects in the protocol caused by the use of a non-reliable protocol as UDP, we had to define some extra headers on each message. These extra fields allow us to control packet losses, acknowledgment losses or even the setting up of timers to manage timeouts. Being more specific, to avoid the problems caused by packet losses we use two mechanisms. On the one hand we use a *sequence number* for each message. This allow us to detect the message losses. On the other hand, we use *timers*. It allow us to resend some messages when we don't get any response during some period of time.

Because there could be retransmissions, we need to implement a mechanism in the *musersd* side to assure that it meets an *at most once* call semantics similar to the *at most once* call semantic used in Remote Procedure Calls (RPC). That's to say, we must avoid that the arrival of several calls for the same command would cause several executions of the same command on the server side. This way we can avoid the appearance of side effects. To avoid this undesired behavior, we have used an "sliding window" mechanism with a window size of one. That's to say, the *musersd* memorize the last request it received and the last response it got to that request. In this way, it can answer directly to that request without executing the command more than once. The request and response messages we have cited before are like those shown in Fig. 2.

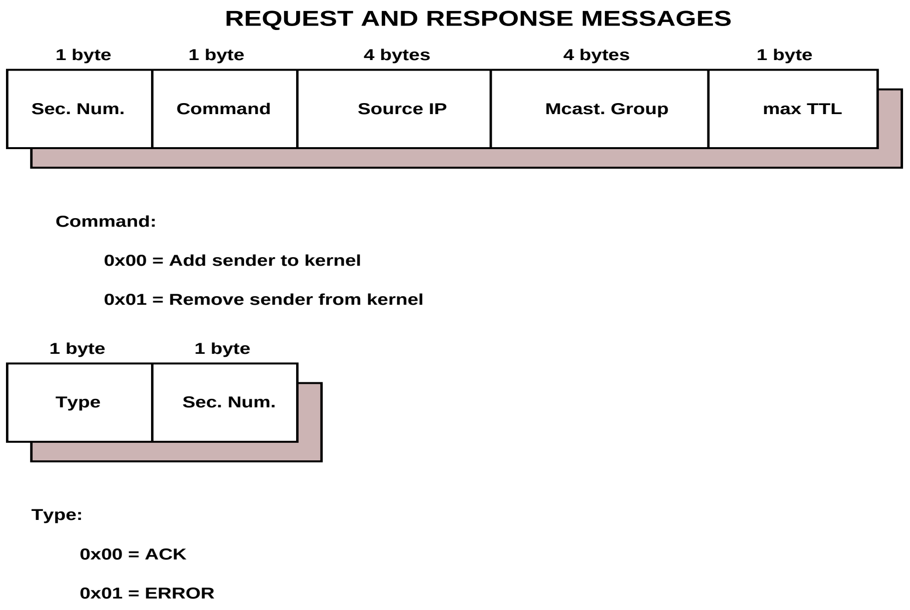

Figure 2: Format of messages.

The benefits that this approach provide are the following:

1. Total independence from MRD.
2. Web-based and totally distributed management. This provides a very easy management. In the same way, supposed our application runs like an applet, we get an additional advantage: platform independence.
3. The only requirement introduced is that the source code of the *mrouter's* OS must be available. All the other hosts taking part in multicast sessions and even that use by the system manager, can run over any other platform. This mechanism requires no changes in final hosts.

This approach has also a drawback. The usage could not seem very intuitive. In addition, the system manager must be looking at every active session using some tool like SDR or mAnnouncer in order to get information about the multicast groups being in use. With that information in mind and using the applet he can add all authorizations to the kernel.

3.4 Final solution

To achieve an easier management, we have decided to create a global tool for the control of MS. This tool allows the system manager to control all multicast active sessions from one window. He can control the advertisement of a new session, the deletion of an active session or even the management of the MS allowed to take part in the sessions.

To achieve our goal we introduce a new element called *mauthserver*. The *mauthserver* acts as a mediator element between the *musersd* an the *mauthclient*. This element has the following functions:

1. It must listen to multicast sessions advertisements. The *mauthserver* implements the SAP(Session Announcement Protocol) and SDP(Session Description Protocol) protocols. Every time a new session is advertised, the *mauthserver* captures that announcement packet and sends it to the *mauthclient*. So, the *mauthclient* can show all active sessions to the manager.

2. It gives some Fault Tolerance(FT) to the system. His strategic position allows it to control all the data flow between the *musersd* and the *mauthclient*. This element will successfully recover the system when any of its extremes falls down. To achieve this recovery, the *mauthserver* will save information from all the sessions and also from all the authorized users. When a crash occurs, the *mauthserver* will retrieve the saved data in order to bring the system to a previous normal state.

A general scheme showing this approach can be seen in Fig. 3. This figure shows a totally decentralized approach. In this approach, there exists some entities communicating among them without imposing extra requirement on the locations of the entities.

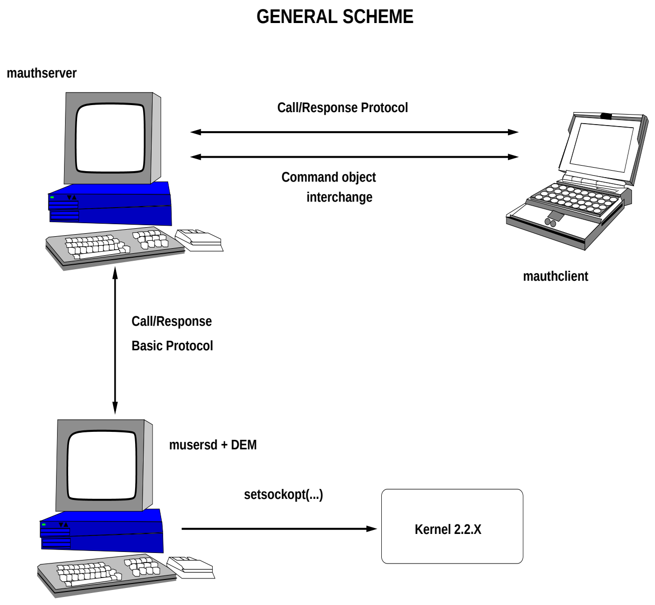

Figure 3: System architecture.

As these entities need to communicate among them, it is necessary to define the communication schemes they must use.

3.4.1 Communication between the *musersd* and the *mauthserver*

The protocol used by this two entities, will be the same call/response mechanism that we have previously defined to communicate *mauthclient* and *mauthserver*.

The *mauthserver* will send commands coming from the *mauthclient* to the *musersd* and then, it will receive responses from the *musersd*. Depending on the response type, the new state will be saved to disk and the responses will be sent to the *mauthclient*.

The protocol and the format of the messages are the same as those in the previous communication scheme.

3.4.2 Communication between the *mauthserver* and the *mauthclient*

For the *mauthserver* and *mauthclient* to communicate each other,we need two communication schemes: on the one hand we need the *mauthserver* to advertise to the *mauthclient* the arrival of a new session announcement or a removal of an existing session deletion. On the other hand, we need the *mauthserver* to accept all the requests made by the manager using the interface offered by *mauthclient*. Valid requests are the registration or the deletion of a MS. This requests will be sent from *mauthclient* to *mauthserver*.

To achieve the latter communication scheme, the solution is to reuse the same protocol we defined in the section above. To achieve the former one we are going to create a new communication protocol.

As the *mauthclient* and the *mauthserver* are implemented using Java, we are going to take advantage of the object serialization facilities offered by this programming language. Every time the *mauthserver* hears a new session advertisement, it will send to the *mauthclient* an object of to the class *Command* using a socket. One instance of the *Command* class has an attribute called *type* (that allow us to identify if we are talking about adding a new session or deleting an existing one) and one other attribute being an instance of the class *Session*. The second attribute contains all the information related with the session which the command refers to. That's to say, multicast groups, media details, title, authorized MS, etc.

In this way, the *mauthclient* can always be aware about active sessions or even the creation or removal of sessions.

There are some advantages when using this approach. They can be seen in the following points:

1. This approach offers an operative and easy-to-use solution. In fact, we have made an integral management tool allowing us to control MS. This tool allows the system manager to establish in an easy, dynamic and Web-based way all the parameters related to everyone of his MS. In this way, the system manager can control every multicast packet leaving of his subnet and this allows him to avoid all those problems commented on the Introduction section.

2. It proposes a totally flexible and distributed approach. This approach is easy to integrate in different environments.

3. The use of the Java programming language in the development of all the entities of the system except the *musersd*, offers us a total platform independence that helps us to get some extra flexibility. In fact, the only element that is not platform independent is the *mrouter*. The only requirement to that element is that the source code of the OS must be available.

4. It simplifies the system manager's tasks. The *mauthclient* has a totally friendly, intuitive and easy-to-use interface. This user-interface is shown in Fig. 4.

5. Of course, this approach continues being independent of the MRD used. In this way, it allows us to easily incorporate our system in different environments and situations.

The main drawback we can see on this approach, it is that we need a manager who centralize all the control process and who establish all the parameters related to every MS. As we could see on the future work, we are planning to avoid the figure of a manager. We are trying to make the registration process as much automatic as possible.

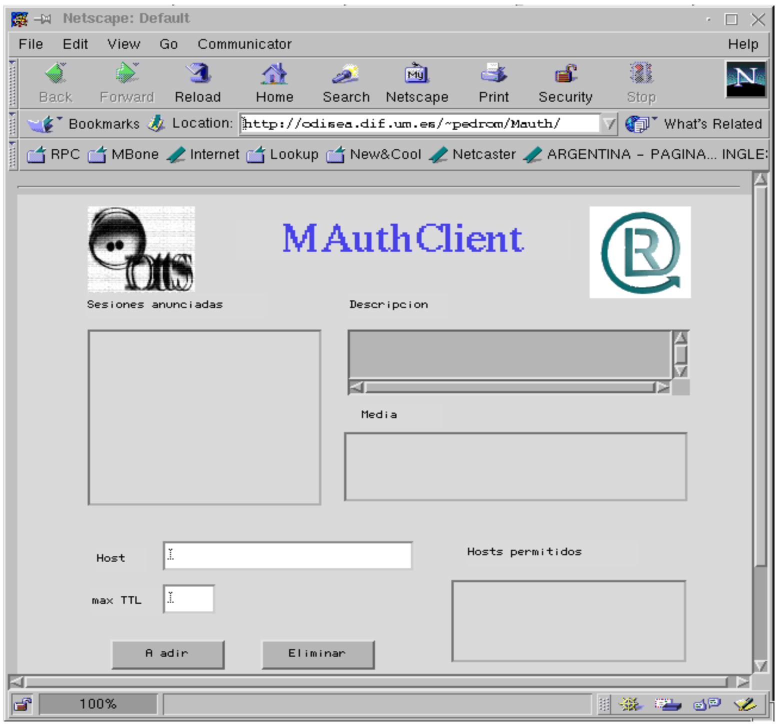
Netscape: Default
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://odisea.dif.um.es/~pedrom/Mauth/
What's Related
RPC MBone Internet Lookup New&Cool Netcaster ARGENTINA - PAGINA... INGLE
MAuthClient
Sesiones anunciadas
Descripcion
Media
Host
max TTL
Hosts permitidos
A adir
Eliminar
100%

Figure 4: Administration applet.

4 Conclusions and future work

The IP Multicasting technology combined with the MBone offers us an perfect environment for video-conferencing over the Internet. But, it can be also used by some other services involving simultaneous communications among more than two hosts in the net.

Despite this, IP multicast has several lack that need to be improved. There is a lot of ongoing work in improving or developing better multicast routing algorithms. But, there have not been sufficient attention to some other issues like controlling users, error monitoring, multicast networks debugging and data privacy.

As we have seen along the paper, we can use several approaches to solve the problem. These approaches range from an application-level solution (allowing nobody to run multicast applications until they get authenticated) to a kernel-level solution based on extending the IGMP protocol.

We have decided to develop a halfway approach. This approach allows us to check that all is working fine before implementing a more complex scheme.

The solution we have developed is totally operative and works as it was supposed to do. That's to say, it allows the system manager to control who gets connected to the MBone, who sends multicast datagrams and the TTL those datagrams have been sent with. The manager defines a maximum TTL for each user and the system drops all multicast datagram sent with a bigger TTL that the assigned to that user. All this process is made in a transparent way for the final user.

In addition, there are no changes on MRD. We only need to add some functionality to the kernel of the *mrouted* host. The changes we need are minimal and only involve the addition of several system calls.

Despite it is a total operational system, this does not means that it couldn't be improved. In fact, we are introducing some of that improvements nowadays.

The first improvement we can made is to setup an authentication mechanism. We must keep in mind that we are talking about communications among different hosts, so we need to setup some mechanism allowing us to identify what hosts are actually participating in the communications process. We need to assure that the multicast datagram comes from the host that it is supposed to come. We are thinking about several possibilities based on the use of encryption. For example, IPsec would be a nice choice.

Nowadays, the user authentication is made by manager. He has to define what users are allowed to send information and what is the maximum TTL a user can use. However, all this system can be mesh into a RADIUS scheme in order to leave the system manager apart from this kind of tasks.

The system main goal is to avoid that spiteful users could clog the communication links by the massive sending of data to certain multicast groups. However, it controls nothing about the information reception. In order to achieve the reception control we need some mechanism to encrypt the data when we work on shared-medium environments.

Acknowledgments

We would want to thank for their collaboration to all the people that helped us while we were looking for a solution or implementing it.

So we want to thank to Norihiro Ishikawa from the NTT Information and Communication Systems Laboratories his help answering some questions related to their internet-draft his hardcopy from the experimental implementation of his system over FreeBSD.

In addition, we would like to thank to Francisco Cruz, Jesús Gonzalez Barahona and his working group in the Carlos III University from Madrid for their interest in hearing of our solution meetings celebrated in the IRIS-MBone working group.

Finally, we thank to the *Comisión Interministerial de Ciencia y Tecnología (CICYT)* for partially supporting this work by means of the project TEL97-2001-E and the project TEL98-0454-C02-01.

References

[1] Vinay Kumar. “MBone: Interactive Multimedia on the Internet”. New Riders, 1996. ISBN 1-56205-397-3.

[2] Steeve Deering. “IP Multicast Extensions for 4.3BSD Unix and related systems”. June 1989. Standford University

[3] D. Waitzman, C. Partridge, S. Deering. “Distance Vector Multicast Routing Protocol”. RFC 1075

[4] W. Fenner. “Internet Group Management Protocol. Version 2”. RFC 2236. November 1997.

[5] David A. Rusling. “The Linux Kernel”. REVIEW, Version 0.8-2. March 1998.

[6] Mark Handley. “SAP: Session Announcement Protocol”. INTERNET-DRAFT. November 1996.

[7] M. Handley, V. Jacobson. “SDP: Session Description Protocol”. RFC 2327. April 1998.

[8] N. Ishikawa, N. Yamanouchi, O. Takahashi. “IGMP Extension for Authentication of IP Multicast Senders and Receivers”. INTERNET-DRAFT. August 1998.