# Secure High Definition Video Conferencing

Master of Science Thesis

Guillem Cabrera Añón

Stockholm / Kista June 2008

---

Telecommunication System Laboratory
KTH Microelectronics and Information Technology

# *Preface*

This job was carried out as part of a final degree project in the Telecommunication Systems Laboratory (TSLab), Royal Institute of Technology (KTH), Stockholm and was developed in the framework of the HDVIPER Celtic project, in collaboration with Fundació i2Cat.

I would like to thank the following people either for their contribution to this study or for the opportunity they gave me to make it:

- ***Erik Eliasson***

  My project advisor, for all the knowledge about SIP and network security he transmitted me. Also for all his practical advice, his stimulating ideas and his help on the C++ implementation using miniSIP libraries.

- ***Björn Pehrson, Jesús Alcober and Flaminio Minerva***

  TSLab director and i2cat HDVIPER coordinators, for giving me the opportunity to work in this project at the TSLab in collaboration with i2cat.

- ***Xavier Calvo and Javier López***

  i2cat HDVIPER developing team, for all their advice on the video conferencing environment and their work in the RTP Packet Reflector.

# *Abstract*

The aim of this review is to study technologies involved in a video conference through Internet. Some security issues and solutions to them are also covered in this report.

At first, several video conference environments are presented to clarify concepts. Also some commercial solutions are mentioned.

Secondly, signalling protocols, specially SIP, are studied to be used in the set up of a video conference. Possibilities to secure SIP are also covered in the theoretical study.

Thirdly, the Secure RTP protocol is presented to be used to protect the media flows. Then a key agreement mechanism, MIKEY, is stated to make the key agreement needed to establish a crypto session for SRTP.

After the background study, an implementation of a secure video conferencing platform using miniSIP and RTP Packet Reflector is proposed.

Then, the final implementation is detailed, showing up the problems appeared during this process and possible solutions to them. Possible new features for the system are also proposed.

Finally, some measurement results taken using the new software are presented and analyzed.

# Table of Contents

# Table of Figures

# 1. Introduction

In a global situation, where companies and institutions operate all over the world and Internet is present in almost all countries, video conferencing is becoming an essential tool to communicate people in very far locations. It is a clear example of how technology can be applied to solve real needs of the society.

Nowadays, video conferencing can be used in business meetings, telemedicine or medical councils and even in distant education. To all these reasons can be added that, since video conferences suppress the need to travel in order to have a meeting, it also contributes to environmental preservation-care.

In order to increase the feeling of the participants in a video conference to be in the same room, new media technologies can be used. High Definition video technologies can improve the user experience, since details can be noticed.

The quick development underwent by Internet technologies in last years increased the bandwidth available for user applications. This fact, linked with the growth of the SIP protocol and the H.323 specifications in the VoIP environment, has allowed the manufacturers to build simple-to-use video conference solutions.

Finally, the use of Internet to transmit the media of a video conference shows some security issues to be solved. Doctors and businessman of any corporation probably would like to be sure only authorized people can watch and listen the conference, so some security measures must be implemented to protect this information.

# 2. Background

This section presents a theoretical study about technologies and products involved in secure High Definition video conferencing.

Firstly, current solutions in video conferencing are analyzed. It is also presented the concept of a centralized solution for multi point conferences.

After that, signalling protocols to control media sessions and media issues (basically video coders) are studied.

Finally, security solutions for media and signalling are discussed, making several proposals to provide different security levels to video conferences.

## *2.1. Video conferencing*

A video conference is usually defined as a meeting of three or more people placed in different locations using any technology that enables them to see and to hear each other almost as if they were in the same room.

The difference between a video conference and a video call is the number of participants in the meeting: two people would be considered just a video call (mainly because it can be done as a peer-to-peer call), whereas three or more people would be considered as a video conference.

A video conference could also include file sharing, white board sharing and instant messages, which helps to have the feeling of being in a real meeting.

Thanks to all these functionalities, video conferencing is a very valuable tool for telemedicine, distance education and business, because it allows to communicate people at different and very far locations without the need to travel.

### 2.1.1. History

The first approach for a video conference was made using two different UHF radio channels to send the signal obtained from television cameras. This system was used in the first space flights by the NASA and some television channels in the 60s and the 70s [1].

Few years after, television channels moved to mobile links to satellites using special trucks for their live connections outdoors. This was a very expensive technology, so it was not common in uses such as education, medicine or business.

For that reason, during the 70s, the American operator AT&T developed the Picturephone, which was a telephone able to send low quality video and a telephone audio channel in a 6Mbps bit rate. Unfortunately, the equipments and the line costs were very high so the system did not succeed [1].

In the 80s, Integrated Services Digital Network (ISDN) links deployed a digital telephony network, so operators started selling few models of videophones. Those devices were used to compress the video and the audio and to transfer that data to another videophone through the digital telephony line [1].

It was in the 90s, with the Internet Protocol (IP) networks deployment and the appearance of new standard based technologies when video calls or video conferencing became popular. Low quality video and audio channels were used as part of most the instant messaging platforms. Also, dedicated and expensive devices were made for big companies, in order to allow their people to meet with no time loose and avoiding travelling costs.

Nowadays, video conferencing is used very often in big corporations, telemedicine and some educational environments. The research efforts in that area are focused on the use of High Definition (HD) video or high quality media, offering secure solutions and the combination of both.

### 2.1.2. Videoconferencing architectures

Depending on the way the media flows are treated and sent, several solutions have been proposed in videoconferencing environments through IP networks.

Videoconferencing seems to be a perfect scenario to use multicast, since media flows generated by one source must be delivered to several destinations (see Figure 1). Unfortunately, in many situations multicast will not work between users so alternative solutions must be considered in case it is not available [2] [3].

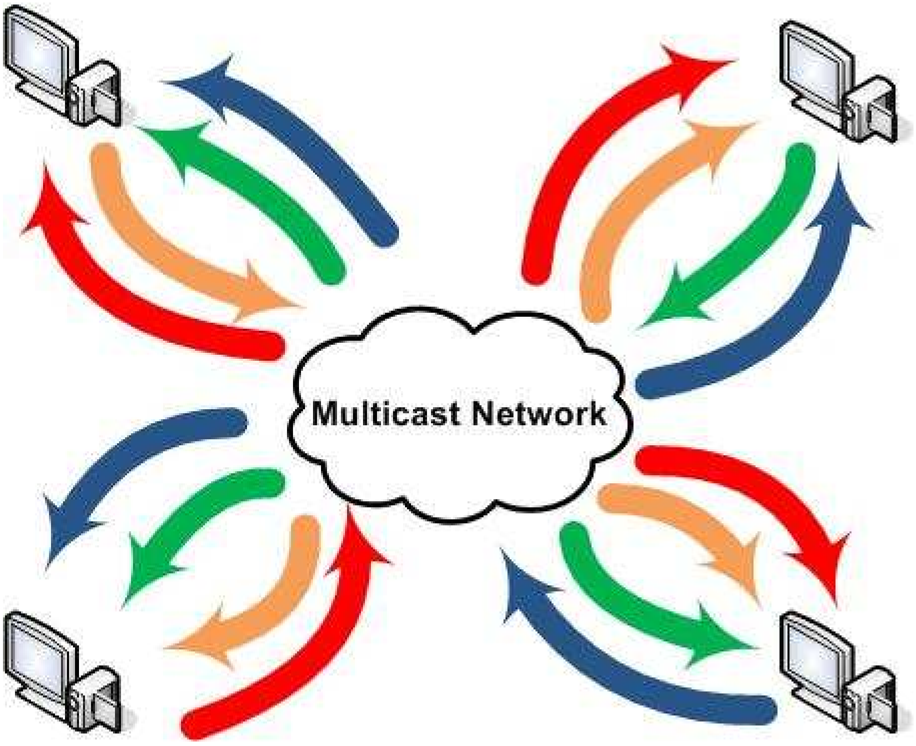


**Figure 1** Video conferencing in a Multicast Network

### *Full Mesh*

In this architecture, each participant in a virtual meeting is sending his or her media flows to all other people in the meeting, as shown in Figure 2. It can also be understood as individual video phone calls between all participants in the conference.

The weakness of this architecture is that each participant would need a lot of bandwidth to send the flows to all other participants. Also, a lot of bandwidth would be needed to receive all media flows coming from other participants. These reasons would limit the number of people on a meeting to the number of flows that can be fit in the Internet link bandwidth.

The biggest strength of this solution is that no centralized architecture is needed, so it is inexpensive, no delays are introduced in the communication and all signalling is done as a peer-to-peer communication between the clients.

It is clear that this could be a good solution for low quality video and audio flows, but it is unaffordable with high quality or High Definition flows because of their high bit rate.

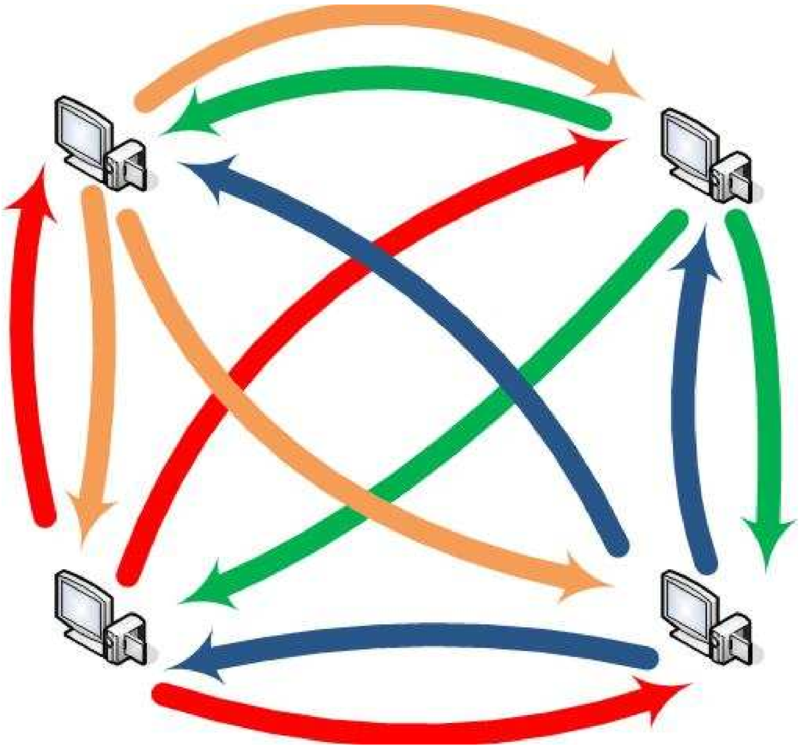


**Figure 2** Full Mesh video conferencing

### *Forwarding*

Another possible solution is to establish a centralized architecture, with a main node receiving all flows and replicating them to all other participants [2] [3] [4]. This central node is called Multipoint Control Unit (MCU) and it simulates a multicast network, since each participant is just sending one flow to the MCU but it could be finally delivered to all other participants. Users could be receiving

several flows too, as if they were in a multicast group with more than one media source.
There are two operational ways to perform the media forwarding, depending if all flows are sent again to all participants or if just one flow is forwarded.

In the first case, the MCU is certainly imitating a multicast network, since the topology will look like as if all participants were in the same multicast group (see Figure 3).

The strength of this possibility is that each participant is only sending his or her media flow once, but it is finally delivered to all other participants. Anyway, all the flows from others must be received, so the download link must be also quite big. However, this situation seems perfect for asymmetric Internet links, such as ADSL.

The biggest weaknesses of this system are the delays added by the central node and that it could become a bottleneck of the system, since it is a centralized element. Nevertheless, the delay can be minimized by using fast software and equipment and high speed lines.

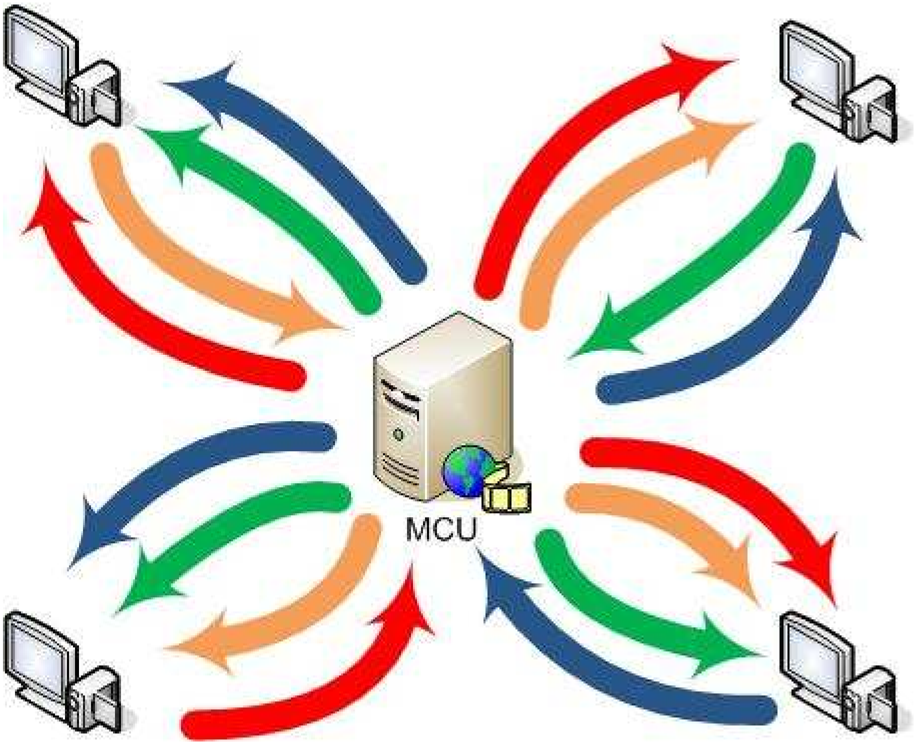


**Figure 3** MCU forwarding all flows

The second possibility using an MCU is to forward only one of the flows received from participants (see Figure 4). This helps to reduce the bandwidth needed to get the flows from other participants, but just allows seeing one of them.

The most practical thing would be that the MCU was able to forward all audio flows and just the video one with the highest volume in the audio channel.

However, this is quite slow because the audio flow must be analyzed, so the delay added by the MCU could be high.
A solution that has been proposed by forwarding-solution developers is to have a meeting director, who is able to switch the media flow forwarded by the MCU. He or she can see and hear media flows coming from all participants, but the other participants just receive the one the director chooses. This is a good solution for chaired meetings with a director giving speech turns.

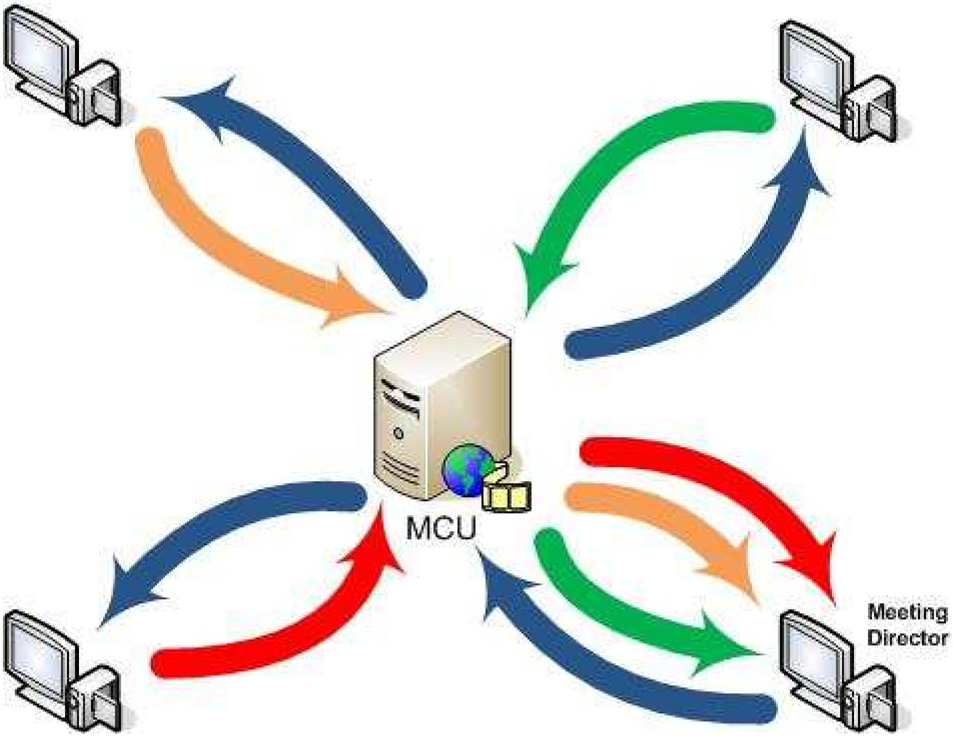


**Figure 4** MCU forwarding one flow

Another smart solution would be to analyze the audio stream and show the one with the highest sound volume.

In some situations, like debates between two people, it would be interesting to forward more than one flow but not all of them. This could be performed the same way is done when forwarding just one.

### *Media mixing*

The last architecture presented is also based on a central node to deal with media flows, but this one is not forwarding media as it comes from the participants.

In these systems, the MCU should be able to create a new media flow compounded of all or some of the flows sent by the people in the meeting (see Figure 5). This will allow all participants to see and to hear each others by receiving just one flow.

However, the mixing process could be very slow and the delays added in the MCU process too long for a video conference.

Moreover, thinking about High Definition conferences, it would make no sense for the users to send HD video flows if the MCU is downscaling them to create a new flow, since details will be lost.

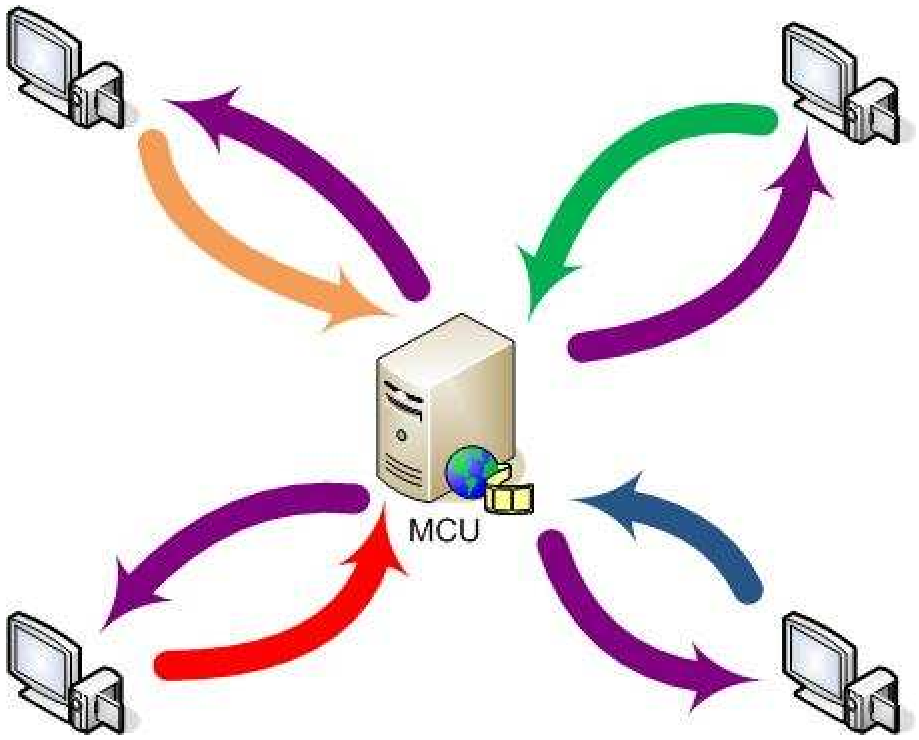


**Figure 5** Media mixing MCU

### 2.1.3. Current solutions

Nowadays, video conferences over IP are very popular, mainly because of the low cost of high speed Internet lines.

On one hand, video phone calls are still made inside of the instant messaging platforms (MSN Messenger, Yahoo! Messenger, AOL IM) or some specific software (NetMeeting, Skype), although some manufacturers are selling IP video phones (D-Link eye2eye videophone).

On the other hand, there are several options for video conference calls. Closed solutions based on specific devices still exist, although most of them are based on open standards for signalling protocols and media transfers (Polycom, Tandberg or Cisco Unified Videoconferencing products). Also, some software solutions are available (NetMeeting).

Even though most of these solutions offer a good security level (some of them use cipher, authentication and integrity standards), none of them is ready to work with high bit rate media flows, such as High Definition flows.

Also, some High Definition systems are present, but they are based on closed and private solutions and often can not work at the same time as secure systems.

## *2.2. Signalling protocols for Internet conferences*

In order to establish a video conference over Internet, a signalling protocol must be used. In this area, lots of solutions have been implemented by manufacturers, but the most interesting ones are those based in open standards.

There are two main standard protocols thought to be used in the establishment of media sessions over an IP network, such as phone calls or conferences: H323 and SIP [5].

### 2.2.1. H.323

*H.323* is a standardized protocol from the *International Telecommunication Union* (ITU). Its first version was released in 1996 and several revisions have been made on it. It was quickly adopted by operators as a signalling protocol used for VoIP and video conferences [5] [6].

Since the companies from the ITU are mostly phone operators and big telecommunications companies, H.323 describes very well defined rules to operate and it is oriented to be used in all kind of networks, like IP, PSTN or ISDN.

However, their main weaknesses are that it is not flexible, it is difficult to provide different services with it and it is a very heavy protocol, what makes it non-scalable.

### 2.2.2. The Session Initiation Protocol (SIP)

The *Session Initiation Protocol* is an application-layer control protocol for creating, modifying and terminating sessions with one or more participants, including multicast sessions [7]. It was defined by the *Internet Engineering Task Force* (IETF) in the RFC 2543 from 1996 and it is used in VoIP solutions and 3GPP mobile technology. It was updated in 2002 in the RFC 3261.

SIP can be delivered over any transport-layer protocol, such as UDP, TCP or SCTP, although it is commonly transported over UDP. It is a text-based protocol, as HTTP, so it can easily be read by humans. This make it a little inefficient in comparison to H.323, but at the same time gives it the possibility to interact with other Internet technologies, like web sites (using SIPlets), and reuse components from other IETF protocols, like HTTP or SMTP [7].

As a signalling protocol, SIP does not care about the media transfers, so it is often used together with RTP/RTCP. However, SIP can also be used in Event Subscription and Notification, Session Mobility and Instant Message solutions.

Whereas H.323 is a very well defined protocol with strong rules, the main strength of SIP is its flexibility and that allows developers to easily design new services using it. Also, the SIP philosophy is to give the clients the intelligence, so the network core can be very simple and the system becomes very scalable [5].

### *SIP Actors*

The *Session Initiation Protocol* is based on the operation of several actors that will exchange messages in order to establish a session. These actors are the ones listed and explained below [2] [5] [7]:

- **User Agent Client:** applications that generates SIP requests
- **User Agent Server:** applications that processes SIP requests and generates SIP responses
- **User Agent:** applications that interact with the final user. They are usually made of a User Agent Client (UAC) to generate requests and a User Agent Server (UAS) to process requests and generate responses.
- **SIP Proxy:** these elements help to route requests and responses to the current location of the receiver of a message
- **SIP Registrar:** these elements are a special type of proxy. They are the responsible to inform the callers for the actual location of the users of that domain, by resolving to a network address. All SIP users in the domain which the registrar is responsible for would have to register on them

### *SIP messages*

SIP is based in the message exchange between the initiator of the session and the receiver or receivers. These messages are divided in a first line, a header and a body (see Figure 6) [7].

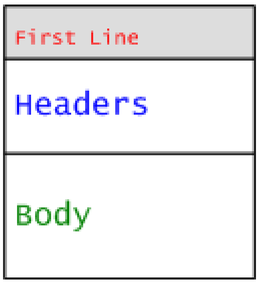


**Figure 6** SIP packet schema

The first line of a SIP message is reserved to indicate the main purpose of the packet, the destination of the message and the protocol version. An example of a first line of a SIP message could be the following:

```
INVITE sip:user1@minisip.org SIP/2.0
```

In this example, the caller would be inviting `user1@minisip.org` to establish a session using version 2 of the SIP protocol.

In the protocol definition, there is a set of SIP methods defined. The most used ones are the following:

- `INVITE` is used to request the start of a session to another user
- `REGISTER` is used to get registered into the client's registrar
- `ACK` is used to acknowledge the receipt of a message
- `BYE` is used to end the session
- `OPTIONS` is used to request client or server capabilities
- `CANCEL` is used to cancel a request
- `INFO` is used to exchange control info once the session is established

Apart from these messages, there is a code defined for the responses to the messages. It is similar to the code defined for responses in HTTP and is based on a number and an optional explanation sentence [7]:

- **Codes 100 to 182:** informational messages, such as `100 Trying` or `180 Ringing`
- **Code 200:** success message, `200 OK`
- **Codes 300 to 302:** redirection messages, like `301 Moved Permanently`
- **Codes 400 to 486:** client error messages, such as `401 Unauthorized`
- **Codes 500 to 505:** server error messages, like `500 Not Implemented`
- **Code 600 to 606:** global failure messages, as `603 Decline`

Also, any message could be added. This is one of the most important features of SIP and what gives its flexibility. By adding new messages, it is possible to perform different behaviours in the User Agents (sending instant messages, sending files, sending XML parameters, …), so then SIP can be used as a signal protocol for lots of applications.

The second parameter in the first line is called the *SIP Uniform Resource Identifier* (SIP-URI). This is the identifier of a user or a device and it is like the phone number in the traditional phone networks or the email address in the email system. Its structure is very similar to email addresses, using @ symbol to separate the user name from the domain name or the network address. The main structure of the SIP-URI would be this:

```
sip:[user:password@]hostname|ipv4addr|ipv6addr[:port;
params]
```

Some examples of SIP-URIs could be the following:

- `sip:user1@minisip.org`
- `sip:user2@212.45.63.24:5060`
- `sip:user:mypassword@147.83.115.46`
- `sip:+34-93-4557644@gateway.minisip.org;user=Dave`

To enable everybody to contact with a SIP-URI, a mechanism to translate a SIP-URI to the current location network address of the user must be performed.

For that reason, every user should register to the SIP registrar of his or her domain.

To find a user given his or her SIP-URI, a client should find first his or her SIP registrar. To achieve this, the process is similar to the one done in SMTP mail exchanges and it usually involves a *Domain Name Server* (DNS) request to get the SRV entry for that domain. This operation can also be done by a SIP proxy on behalf of the original client.

After the first line of the SIP packet, a structured header of the packet is enclosed. It is very similar to HTTP or SMTP headers and it describes the caller, the callee, the path and the message type. There are 37 different headers and they can be divided into 4 main groups [5] [7]:

- **General headers:** used both in request and response messages
- **Request headers:** used in request messages to add extra information
- **Response headers:** used in response messages to add extra information
- **Entity headers:** indicates the length and the content type in the body of the message

The most important and most used headers are explained below:

- **Via:** indicates the path (SIP proxies) the message took to arrive to the destination
- **Max-Forwards:** indicates the number of hops the message can do before to arrive to destination. Each proxy server that forwards this message would decrease by one this parameter
- **To:** indicates the name and the SIP-URI of the destination of the message
- **From:** indicates the name and the SIP-URI of the source of the message
- **Call-ID:** unique identification number of the session
- **Command-sequence:** sequence number of the message and SIP method
    - Incremented by one for each new message
    - The same number for the responses to a message
- **Subject:** indicates the subject of the session established
- **Contact:** indicates the SIP-URI to contact directly the source of the message
- **Content-type:** indicates the type of content in the message body
- **Content-length:** indicates the length of the body of the message
- **Accept:** indicates if the message body is accepted by the client
- **Accept language:** indicates the preferred language of the user to receive messages
- **Date:** time of the first request
- **Encryption:** indicates the preferred algorithm to encrypt the body of the messages
- **Record-route:** used by proxies to demand all messages to go through it

- **Require:** indicates the functionalities the client needs to be performed to accept the session
- **Supported:** indicates the features supported by a server
- **Timestamp:** used by the clients to calculate the round trip time delay

Finally, a SIP message can also include a body. It is optional and it usually carries information about the session established or non-SIP parameters to be exchanged.

### *Session Description Protocol (SDP)*

In order to use SIP to establish a media session (i.e. a phone call), it is necessary to agree upon parameters for the session. The caller and the callee must agree which CODEC and which protocols to use, where to send the media flows and where to receive them.

These parameters are not agreed in the SIP message exchange, so another protocol should be used in the SIP message body to do so (see Figure 7). This protocol is the *Session Description Protocol* (SDP), which is another IETF protocol designed to define media session initialization parameters [8] [9].

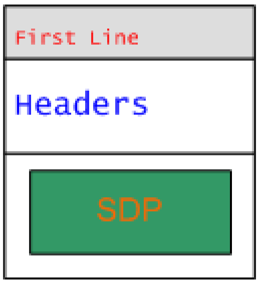


**Figure 7** SIP packet with SDP

Using SDP, participants can define information related to the type of media (audio, video, …), the CODEC (MPEG, PCM, GSM, …), the transport protocol (UDP, RTP, SCTP, …) and port numbers.

Nowadays, the protocol is described in the RFC 4566 and it defines the following list of attributes to define media sessions:

<u>Session description</u>

- **v:** protocol version
- **o:** originator and session identifier
- **s:** session name
- **i:** extra session information
- **u:** URI of description
- **e:** email address
- **p:** phone number
- **c:** connection information
- **b:** bandwidth information
- **z:** time zone adjustments
- **k:** encryption key

- **a:** session attribute

Time description
- **t:** time the session is active
- **r:** repeat times

Media description
- **m:** media name and transport address
- **i:** media title
- **c:** connection information
- **b:** bandwidth information
- **k:** encryption key
- **a:** media attributes

### *SIP dialog*

As said above, SIP is a protocol based in the message exchange between SIP User Agents in the clients. Attending to this and taking into account all explained about SIP, an example of a SIP dialog to establish a video phone call is presented.

Imagine two users, Alice and Bob, with their SIP-URIs, `alice@kth.se` and `bob@mnisip.org`. Alice wants to make a video phone call to Bob and she knows about Bob's SIP-URI but not about his IP address, his location nor his SIP Proxy Registrar. The dialog to establish a call will be the one shown in Figure 8.

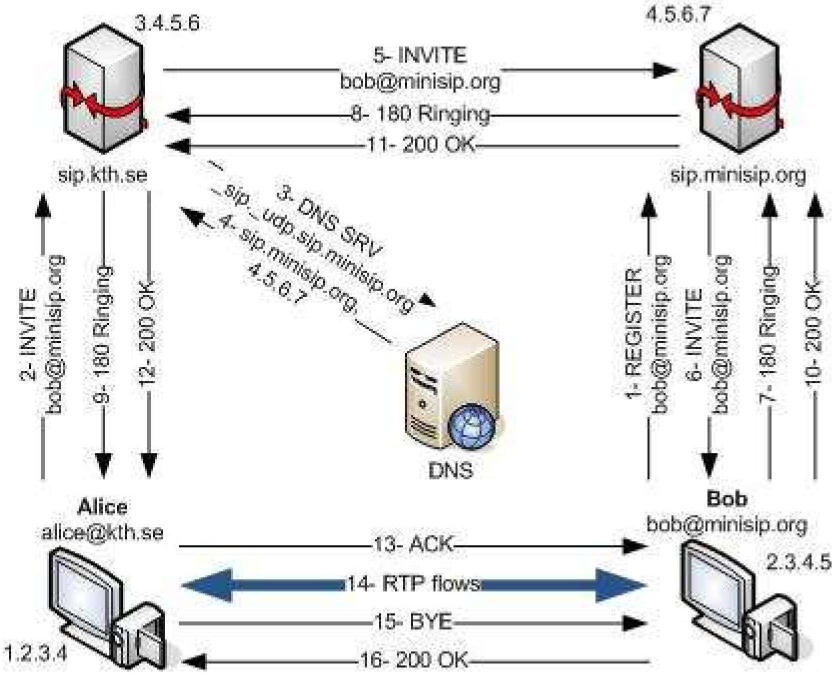


**Figure 8** Video phone call establishment SIP dialog

The messages in Figure 8 would be the following:

- **Message 1:** Bob is registering in the SIP Proxy Registrar responsible for the domain `minisip.org`, which is `sip.minisip.org`. The content of the SIP message would be this:

```
REGISTER sip:sip.minisip.org SIP/2.0
Via: SIP/2.0/UDP 2.3.4.5:5060
From: sip:bob@minisip.org
Contact: Bob <sip:bob@2.3.4.5:5060>
CSeq: 1 REGISTER
Call-ID: 2e45f4678@2.3.4.5
Expires: 7200
```

- **Message 2:** when Alice wants to contact Bob, she sends an INVITE message through her SIP Proxy. This message contains an SDP message in the body, in order to offer Bob several options to establish the media session.

```
INVITE sip:bob@minisip.org SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4:5060
From: sip:alice@kth.se
To: sip:bob@minisip.org
Call-ID: 2023@1.2.3.4
CSeq: 1 INVITE
Content-type: application/sdp
v=0
s=Video phone call
e=alice@kth.se
c=IN IP4 1.2.3.4
t=0 0
m=audio 10000 RTP/AVP 0 8
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
m=video 10001 RTP/AVP 34
a=rtpmap:34 H263/90000
```

In this SDP, Alice is offering Bob to use either PCMU or PCMA audio codec in her UDP port 10000 over RTP and H263 video codec in her UDP port 10001 also over RTP for the video phone call.

- **Message 3:** since Alice SIP proxy (`sip.kth.se`) does not know about Bob's SIP Proxy, it makes a DNS Request in order to get its address. The proxy would ask for the SRV entry and it would indicate to find a SIP and UDP capable server.

- **Message 4:** the DNS response would include the name of the SIP proxy for `minisip.org` domain as well as its IP address.

- **Message 5:** now, `sip.kth.se` can forward the Alice `INVITE` message to `sip.minisip.org`, adding a Via header in the SIP header:

```
INVITE sip:bob@minisip.org SIP/2.0
Via: SIP/2.0/UDP sip.kth.se:5060
Via: SIP/2.0/UDP 1.2.3.4:5060
From: sip:alice@kth.se
To: sip:bob@minisip.org
Call-ID: 2023@1.2.3.4
CSeq: 1 INVITE
Content-type: application/sdp
v=0
s=Video phone call
e=alice@kth.se
c=IN IP4 1.2.3.4
t=0 0
m=audio 10000 RTP/AVP 0 8
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
m=video 10001 RTP/AVP 34
a=rtpmap:34 H263/90000
```

- **Message 6:** the `INVITE` message would be finally delivered to Bob by his SIP Proxy, adding another Via header:

```
INVITE sip:bob@minisip.org SIP/2.0
Via: SIP/2.0/UDP sip.kth.se:5060
Via: SIP/2.0/UDP 1.2.3.4:5060
From: sip:alice@kth.se
To: sip:bob@minisip.org
Call-ID: 2023@1.2.3.4
Cseq: 1 INVITE
Content-type: application/sdp
v=0
s=Video phone call
e=alice@kth.se
c=IN IP4 1.2.3.4
t=0 0
m=audio 10000 RTP/AVP 0 8
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
m=video 10001 RTP/AVP 34
a=rtpmap:34 H263/90000
```

- **Messages 7, 8 and 9:** Bob's User Agent would send a `180 Ringing` to Alice in order to inform her User Agent the `INVITE` message was received. This message is going to be forwarded back to Alice through the SIP Proxies, each one of them adding a Via Header.

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 2.3.4.5:5060
From: sip:alice@kth.se
To: sip:bob@minisip.org
Call-ID: 2023@1.2.3.4
Cseq: 1 INVITE
Content-length:0
```

- **Messages 10, 11 and 12:** as soon as Bob picks up the phone, his User Agent would send a `200 OK` response to Alice. This message is also going to be delivered through the proxies, since Bob does not know yet Alice's address.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 2.3.4.5:5060
From: sip:bob@minisip.org
To:sip:alice@kth.se
Call-ID: 2023@1.2.3.4
Cseq: 1 INVITE
Content-type: application/sdp
v=0
s=Video phone call
e=bob@minisip.org
c=IN IP4 2.3.4.5
t=0 0
m=audio 20000 RTP/AVP 0 8
a=rtpmap:8 PCMA/8000
m=video 20500 RTP/AVP 34
a=rtpmap:34 H263/90000
```

In the SDP message, Bob chooses to use PCMA over RTP in his 20000 UDP port for the audio flow and H263 over RTP in his 20500 UDP port for the video flow.

- **Message 13:** when Alice gets the `200 OK` message from Bob, she sends an acknowledgement for this message to tell Bob she accepts the parameters Bob chose. At this moment and thanks to the Via headers, Alice knows Bob's location, so this message is sent directly to Bob.

```
ACK sip:bob@minisip.org SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4:5060
From: sip:alice@kth.se
To: sip:bob@minisip.org
Call-ID: 2023@1.2.3.4
Cseq: 1 ACK
Content-length: 0
```

If Alice did not accept Bob's SDP offer (either for the CODEC, the port or anything else), she would send a `CANCEL` message instead. Then, the connection would be closed.

- **Message 14:** Alice and Bob will be sending each other 2 RTP flows containing audio and video information. They should use the CODECs and the ports they agreed in the SIP dialog, thanks to the SDP protocol.

- **Message 15:** when the video phone call ends, Alice would hang up and media flows would stop sending. At this moment, her User Agent is going to send a `BYE` message to Bob's User Agent, so it notices the call is over.

```
BYE sip:bob@minisip.org SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4:5060
From: sip:alice@kth.se
To: sip:bob@minisip.org
Call-ID: 2023@1.2.3.4
Cseq: 345 BYE
Content-length: 0
```

- **Message 16:** finally, Bob would accept the end of the call by sending a `200 OK` message to Alice.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 2.3.4.5:5060
From: sip:bob@minisip.org
To: sip:alice@kth.se
Call-ID: 2023@1.2.3.4
Cseq: 345 BYE
Content-length: 0
```

## *2.3. Dealing with media in video conference environments*

Once the signalling protocol agrees the parameters to establish a media session, data flows containing audio, video or other kind of information can be sent by participants.

However, it is necessary to define some protocols to allow participants to read correctly these media flows.

### 2.3.1. Real-time Transport Protocol (RTP)

The *Real-time Transport Protocol* (RTP) is a standardized packet format designed to deliver media contents (mostly video and audio, but also images) over an IP network [10]. It was first defined in 1996 as RFC 1889, which was replaced by RFC 3550 in 2003.

It was originally thought as an add-on for the UDP protocol in media environments, although it can be also delivered over a TCP layer. It has no defined port to be delivered, so it is usually delivered in the wide area of non-defined ports. However, it is said in the standard that RTP must use an even port and the *Real-time Transport Control Protocol* (RTCP) use the next odd port.

An RTP packet is divided into a 12-bytes header and a payload (see Figure 9). The payload would carry the data of the packet and in the header the following information is provided [5]:

- **RTP Version [V]** (2 bits)**:** indicates the version of the protocol. Current version is 2
- **Padding [P]** (1 bit)**:** indicates if there is extra information at the end of the packet, in order to exactly fit on an 8 bit alignment
- **Extension [X]** (1 bit)**:** indicates if any extension of the protocol is used in the packet
- **Contributors [NCSRC]** (2 bits)**:** indicates the number of contributors to that session. It is used to read the CSRC optional headers
- **Marker bit [N]** (1 bit)**:** it is used to inform the application that something special is in that packet. Usually is always off in audio streams and 1 in video packets containing the last information of a frame
- **Payload type [PT]** (7 bits)**:** identifier of the type of data in the payload of the packet. Usually it is an identifier of the codec used in the media transported, defined in the RFC 3551
- **Sequence number [SN]** (8 bits)**:** sequence numbers increased by one at each packet
- **Timestamp [TMP]** (16 bits)**:** time of the acquisition of the first sample carried by the packet
- **Synchronization source identifier [SSRC]** (16 bits)**:** identifier of the source of the packet
- **Contributing source identifier [CSRC]** (16 bits)**:** identifier of the contributors to the media session. It is optional and it will be one for each participant when the media goes through a mixer

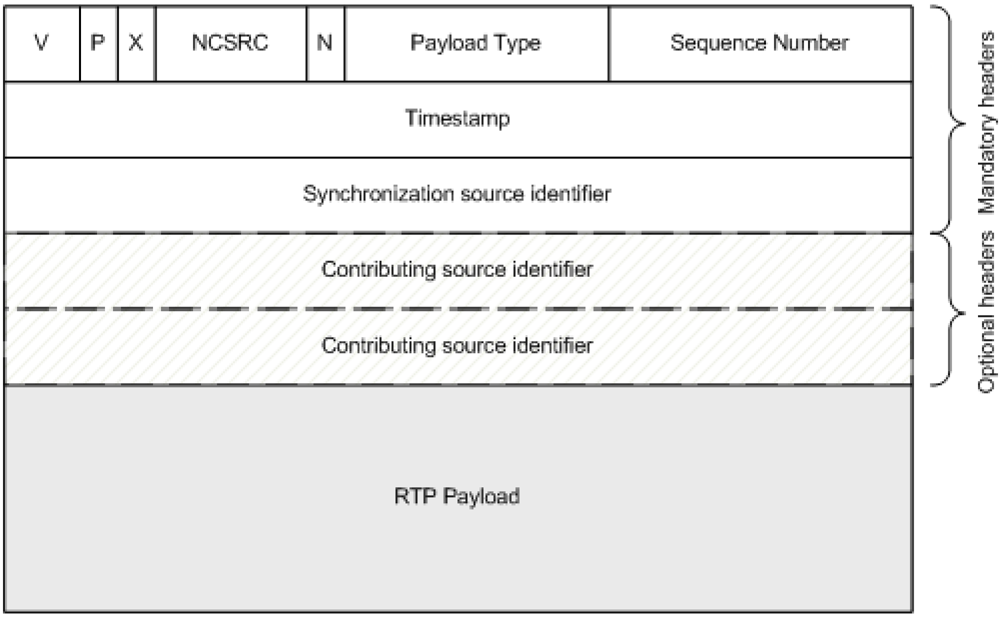

V
P
X
NCSRC
N
Payload Type
Sequence Number
Timestamp
Synchronization source identifier
Contributing source identifier
Contributing source identifier
RTP Payload
Mandatory headers
Optional headers

**Figure 9** RTP packet schema

The RTCP protocol was designed as a brother protocol of RTP in the same RFC 3550. It is usually transmitted periodically together with an RTP flow and provides quality of service feedback and control information to the source and the receiver of the media flow, like timestamps linking to *Network Time Protocol* (NTP) times or changing CODEC requests.

### 2.3.2. Delays and Video coders

In a video conference, video flows should be treated in a special way. In these environments the delay must be as small as possible, in order to give the participants the feeling that they are in front of each other.

Attending to the interactivity ITU recommendation, the maximum delay between each part should not be longer than 150ms (Round Trip Time <= 300ms). This means that a participant should see and listen what happened in the other participants' rooms 150ms before. This is really difficult to achieve with nowadays technologies if one consider using HD video, but gives the idea on how the delay should be considered.

To measure the delay, several times must be taken into account: acquisition time, compression time, packetizing time, network transmission time, unpacketizing time, decompression time and jitter-control display buffer time (Figure 10).

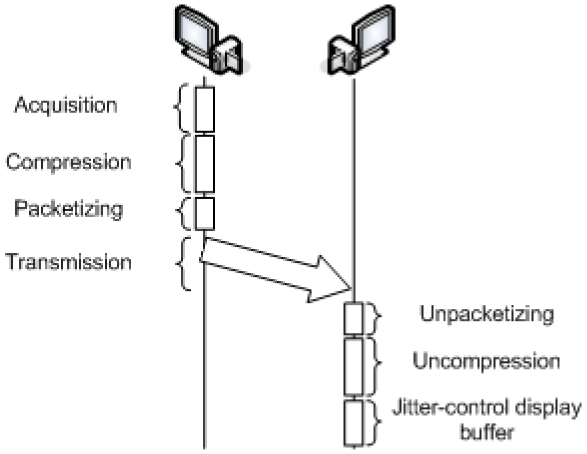


**Figure 10** Video conferencing delays

Regarding to this time schema, it is easy to realize that the less is the compression/decompression time, the less is the total delay of the system. So then, it is a good idea to use media CODECs that do not take too much time to compress media or even not to compress it, always trying to compromise compression time and network transmission time.

#### *Digital Video (DV)*

This format of video is the one used by most of the domestic video cameras. It consists in raw PAL/NTSC images usually obtained through the Firewire interface (IEEE 1394). It mixes in a single 25Mbps flow video and audio signals.

Since no compression is applied and the output bit rate seems quite reduced, it could be valid for a first approach in video conferencing. However, having the audio and video flows mixed could be a problem if the sound is the parameter analyzed to decide which flow is being showed. Moreover, the quality of the video flow is just PAL/NTSC, which means is no High Definition.

#### *High Definition Digital Video (HDV)*

This format is the High-Definition evolution of the Digital Video format. It offers the possibility to fit in a 25Mbps flow an audio and a HD video flow [12]. However, MPEG-2 compression is applied, so the delay is increased.

Because of the compression introduced by HDV cameras, this seems not to be a right format for High-Definition video conferencing.

#### *MPEG4/H.264*

The MPEG4 or H.264 codec is the newest video coder from the *Moving Pictures Experts Group* (MPEG) and the ITU. It uses several techniques to compress the image flow, such as backward and forward movement prediction, wavelet transform (like JPEG200 standard) and adaptive quantifiers.

These techniques make possible to obtain high-quality results with a contained bit rate, which is what is looked for High Definition video conferencing.

This standard was designed to be delivered through a packet network, so it was defined also as an RTP encapsulation for it in order to improve the efficiency.

Most of the current commercial video conferencing solutions offered by manufacturers support and take advantage of this standard, usually implementing a proprietary profile of it. However, and specially because of the delay introduced by the forward movement prediction used in most of H.264 implementations, the delay introduced when using them is long enough not to be used in video conferencing environments.

Several studies are being done in this area at the moment, most of them focused on reducing the compression time avoiding the forward movement prediction.

## *2.4. Security issues*

Until this part of the document, several protocols and solutions have been studied. However, no security aspects have been taken into account.

There are several ways to secure these protocols and different security levels for each one, so they must be analyzed.

### 2.4.1. Security definitions

First of all, it is necessary to define what means that the different subjects that can be secured in a communication through a packet network.

### Confidentiality

In a packet switched network and especially in Internet, there is no possibility to control the path our packets will take. This means that anybody could be listening on the network and looking at the information inside our packets or the packets sent to us.

Confidentiality means that only authorized people can read the information in a message. This is usually achieved by encrypting all the information transferred through the network.

### Integrity

As said above, there is no chance to control our packets along their path to the destination. Anybody could modify the information on them and the receiver would not notice.

To avoid that, some procedures must be done in order to ensure that the receiver is getting exactly what the sender sent, without modifications. This is often done by hash codes of some information the sender and the receiver know and some information in the message.

### Authentication and non-repudiation

Authentication is needed in order to make sure somebody is who he or she claims to be. It is absolutely necessary to be sure the receiver of the information is the right one to provide full-secured communications.

Usually, authentication is merged with non-repudiation. This means that a party in a dispute cannot refuse he or she did something

Both properties are usually achieved by a digital signature procedure, using a *Public Key Infrastructures* (PKI), and also provide integrity.

### Availability

It is important for a service to be available for as much time as possible. In order to avoid down times, it is necessary to think about different techniques to prevent Denial of Service (DoS) attacks.

### Public Key Infrastructure

A Public Key Infrastructure is a set of technologies, policies and procedures that allow the secure use of cryptographic methods, such as ciphering or digital signature [13].

Usually, a PKI is based on the operation of one or more Certificate Authorities (CAs). These authorities are responsible of several user certificates and can assure to anybody that who is using a certificate is who he or she says. However, it is necessary to trust the CA.

A certificate contains a pair of public-private keys that can be used in asymmetrical cryptography procedures, like digital signature or data authentication.

### 2.4.2. Securing SIP

Although of its diverse advantages, SIP shows a severe disadvantage since it does not provide any security measure to secure the protocol itself. To do so, it relies on other generic protocols, such as *Transport Layer Security* (TLS) or *IP-Security* (IPSec) [7].

Several things can be protected in a SIP communication, so it is important to offer solutions for all the issues.

#### *Securing communications*

As seen in the SIP dialog above, SIP was designed as a non-encrypted protocol, so its messages are transmitted as plain text. This is a very important security problem because anybody could see user activities, such as in or out calls or instant messages sent using SIP. To solve this, SIP offers the solution to use TLS, establishing hop by hop TLS links between all nodes in the dialog [14] [17].

In order to require this security measure, a new SIP-URI has been created, the SIPS-URI. The format of the new address is the one below:

```
sips:[user:password@]hostname|ipv4addr|ipv6addr[:port
;params]
```

By using a SIPS-URI, intermediate nodes will be forced to establish secure links between them using TLS [7]. All connections will be then secured, except the one between the destination proxy and the destination user, which could be secured or not depending on the policy of the SIP proxy [14]. If one of the hopes can not be secured, the connection should fail.

Using the example of the SIP dialog above, the blue areas over SIP messages in Figure 11 would be the TLS secured connections:

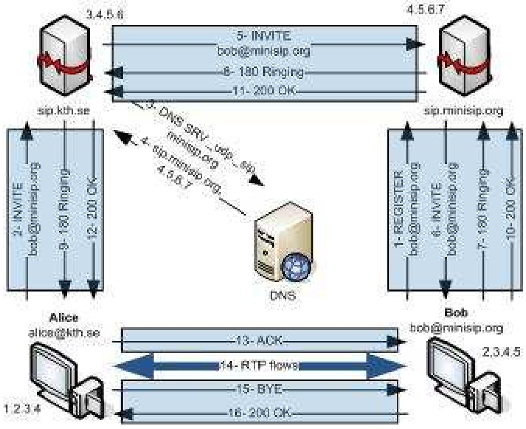


**Figure 11** Secure SIP dialog using TLS

Nevertheless, before creating the TLS links between the proxies, they will need to authenticate each other, so a PKI will be needed. Since it could be useful to provide security in other aspects of video conferencing, the use of a PKI seems to be a logical solution.

### Securing register

When a user is registering to his or her SIP Registrar, some security measures must be taken into account. The SIP Registrar must assure the user is who he or she claims to be by authenticating them before they can register.

To do so, SIP offers two authentication schemas, similar to the ones offered by HTTP:

- **Basic authentication:** this schema is very weak because it is based on sending as clear text user and password. For this reason, it has been deprecated.

- **Digest challenge:** the user tries to get registered in the SIP Registrar with no authentication and the server denies the request and attaches a nonce to the `CANCEL` message [7]. Then, the user would make another REGISTER request, by adding to the packet the result of the MD5 hash function over the nonce, the user and the password. If everything goes right, the user gets registered (see Figure 12).

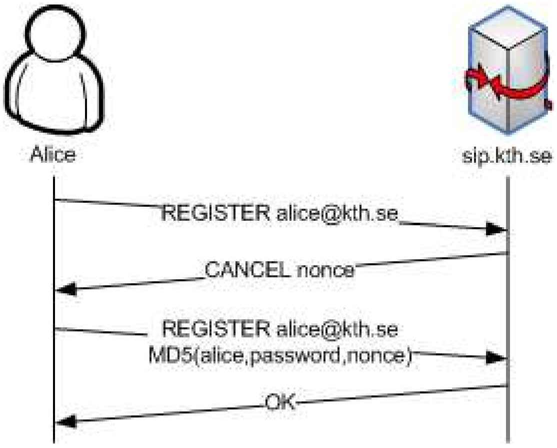


**Figure 12** SIP digest challenge authentication

Though it seems the digest challenge method could be enough, the user is not authenticating the server, so a fake server could impersonate him or her. The solution for that is again the use of a PKI to authenticate the client and the server and the TLS protocol to protect network communications.

### Securing the body message

Another possibility would be to secure only the body of the SIP messages. This will not protect the signalling information but SDP messages or any other information contained in the body message (like instant messenger texts) could be at least protected.

SIP specifies the use of the *Secure/Multipurpose Internet Mail Extension* (S/MIME) to provide confidentiality, integrity and authentication to the body of SIP messages by using a PKI [13] [14] [15].

### *Securing the whole SIP message*

By using S/MIME it is also possible to protect the whole SIP packet, including the headers. This technique copies the whole SIP message (headers and body) inside the body (Figure 13) [16] [17].

Then, S/MIME is used to provide confidentiality, integrity and authentication to the body of the SIP message, which contains the whole original packet. The receiver of the message should decrypt the body and compare the copied message inside with the whole packet.

If both messages are equal, no changes had been made on them. However, if not, some changes had been made in the packet.

This system should take into account that some proxies could add headers to the SIP message, like Via headers. This makes it difficult to be implemented and adds computational complexity to the SIP message dealing process.

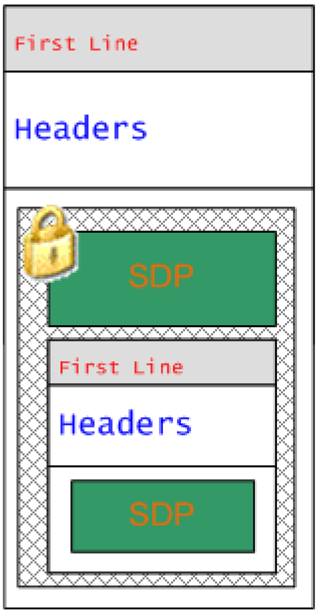


**Figure 13** Securing the whole SIP message

Another problem of this procedure is that the resultant UDP message would probably be bigger than the network MTU so UDP packets will be fragmented. This is a problem because the standard configuration of the most spread SIP proxy (SIP Express Router) [26] discards all fragmented packets. A solution for that could be using a reliable transport protocol (such as TCP or TLS) to deliver these messages, as the standard says.

### 2.4.3. Securing Media: Secure RTP (SRTP)

There are several options to secure the content of the media flows, depending of which level of the TCP/IP stack is protected.

One option is to use IPSec tunnels between participants [18], which is a very heavy solution because it is protecting the network layer. Another possibility is to use Secure RTP to protect the RTP flows carrying media, which is a smart solution for communications over Internet because only application layer is protected.

Attending to previous studies in this area [16] [18], the use of SRTP seems more convenient for video conferencing environments, since it is a close environment to VoIP.

The *Secure Real-time Transport Protocol* (SRTP) is a profile to secure the RTP media transport protocol specified by the proposed standard in the RFC 3711 [19].

The protocol is thought to provide confidentiality, authentication and integrity to the packets of an RTP flow, although only encryption is mandatory. The security is applied to the packet as Figure 14 shows.

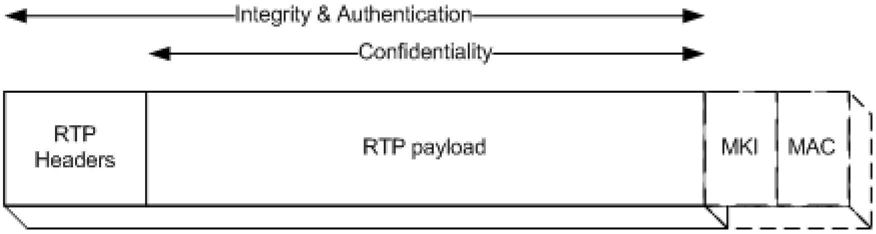


**Figure 14** SRTP schema

Several security issues, such as confidentiality, authentication and integrity, can be solved by using SRTP to protect RTP media flows.

#### *Confidentiality: AES*

In the SRTP draft, confidentiality is achieved by encrypting only the payload of the RTP packets. The encryption is provided by the *Advanced Encryption Standard* (AES) algorithm used in the cipher-stream mode, instead of the block cipher performed usually by AES [13].

The use of AES instead of some other algorithms is not a coincidence. The main reason to choose AES in front of other algorithms is its low computational requirements and that it is often implemented in hardware. These characteristics allow this algorithm to be used in low computational devices [14], such as mobile phones or PDAs, and give it the possibility to encrypt high bit rate flows using powerful machines.

AES describes three different procedures to work under the cipher-stream mode, but only two of them are used in SRTP:

- **NULL cipher:** it can be used to disable encryption, when authentication and integrity protection are needed, but not confidentiality.

  The operation of this mode is very simple: it just copies the input to the output, with no changes in the payload of the RTP packet.

- **Segmented Integer Counter Mode:** this mode is based in typical counter mode behaviour, using one-time-pad keys to cipher the plaintext. The main advantage of this mode is that one-time-pad keys can be pre-computed and then used in the XOR with the plaintext [13].

  This is the default algorithm for SRTP and it uses a 128 bits default encryption key and a 112 bits default session salt key.

  In the below schema (Figure 15), it is shown how the default AES Block Cipher is used in counter-mode. The Initialization Vector (IV in the figure) depends on the source identifier (SSRC), the packet index (Sequence Number) and the salting key.

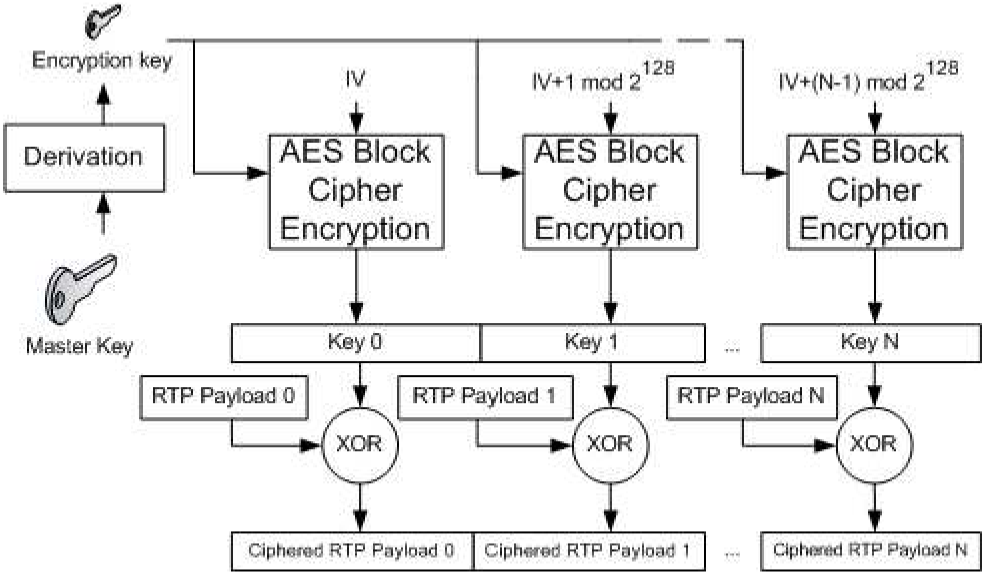


**Figure 15** AES counter-mode

### *Authentication and Integrity: HMAC*

Although the RTP packets were ciphered, anybody in the network could manipulate them or even replace them. This could be a big security hole, because reply attacks can be done against a receiver, making impossible to him or her to receive any media.

To solve these issues, authentication and integrity protection must be provided. This is achieved by a *keyed-Hash Message Authentication Code* (HMAC) based on a *Security Hash Algorithm* 1 (SHA-1) hashing function. This function returns a *Message Authentication Code* (MAC), which is added to the RTP packet with a ciphered payload.

A keyed-Hash Message Authentication code is obtained applying a hash function to the content to be protected, an authentication key and two defined padding (opad, ipad) [13]. Note that the hash is applied after the encryption is done and it protects also the RTP headers.

The default authentication key should be 128 bits and the HMAC-SHA-1 algorithm will generate a 160-bit hash code. In SRTP this hash code is truncated to the last 32 bits (4 bytes) [19], in order not to add too much overhead to the media flow (Figure 16).

Finally, a 4-byte Master Key Identifier is also added to the packet, in order to inform the receiver which crypto context (encryption key, authentication key and salt key) was used to protect that packet (Figure 16).

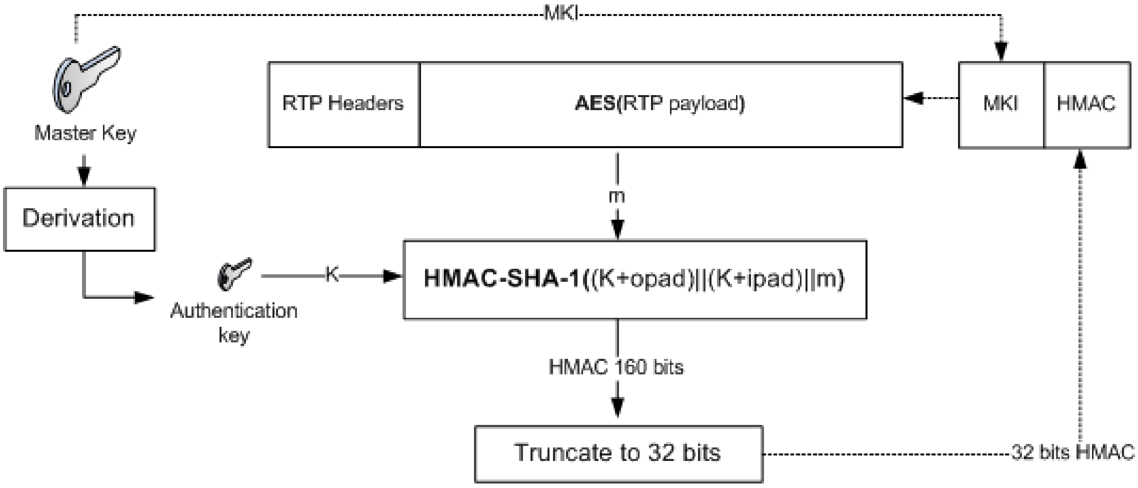


**Figure 16** SRTP HMAC generation

### *SRTP cryptographic issues*

As stated, SRTP uses a master key to obtain the other keys used to protect the RTP packets, but the draft says nothing about how to share this master key between the source and the destination.

In order to get an encryption key, an authentication key and a salt key from the same master key, a derivation function is used to generate this crypto context (Figure 17) [19]. By this way, only a master key needs to be exchanged.

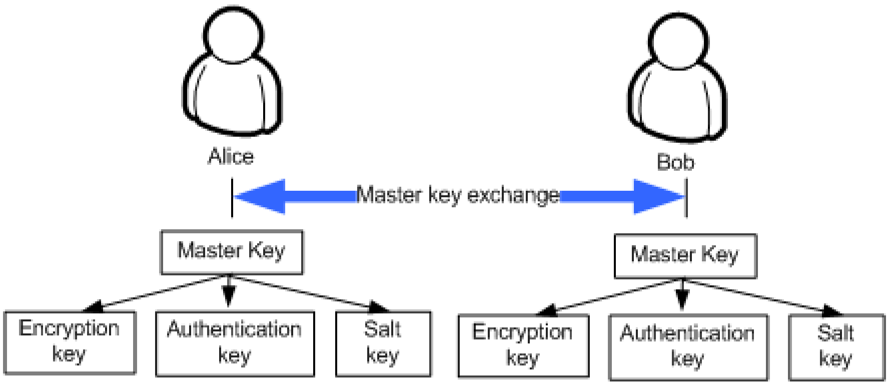


**Figure 17** SRTP crypto context creation

Moreover, SRTP could become insecure if the same key is used to cipher more than $2^{48}$ packets, a number fixed because of the rollover counter and the sequence number lengths. With this amount of ciphered data, the possible attacker could obtain information and break the cryptographic session. In order to avoid this, a re-keying protocol to generate a new crypt context is needed too.

### 2.4.4. Key Agreement: MIKEY

There are some existing key agreement implementations, such as the *Internet Security Association and Key Management Protocol* (ISAKMP) framework and the *Internet Key Exchange* (IKE) protocol [14]. Both of them provide several levels of security.

However, there is a very specific option for protecting multimedia exchanges: the *Multimedia Internet Keying* (MIKEY) [20]. It is been revealed as the best option when using SIP to establish secured multimedia sessions [14] [18].

The purpose of MIKEY is to exchange cryptographic parameters in the media negotiation process. This, when using SDP over SIP to establish media sessions, means that MIKEY should be able to perform a mutual authentication and agree a master key in just one roundtrip.

MIKEY defines a way to exchange a master key, called Transport Generation Key (TGK). From this TGK, MIKEY is able to get a Transport Encryption Key (TEK), which will be the one used as the SRTP master key (see Figure 18) [14] [20]. The protocol also provides a mechanism to exchange other security parameters.

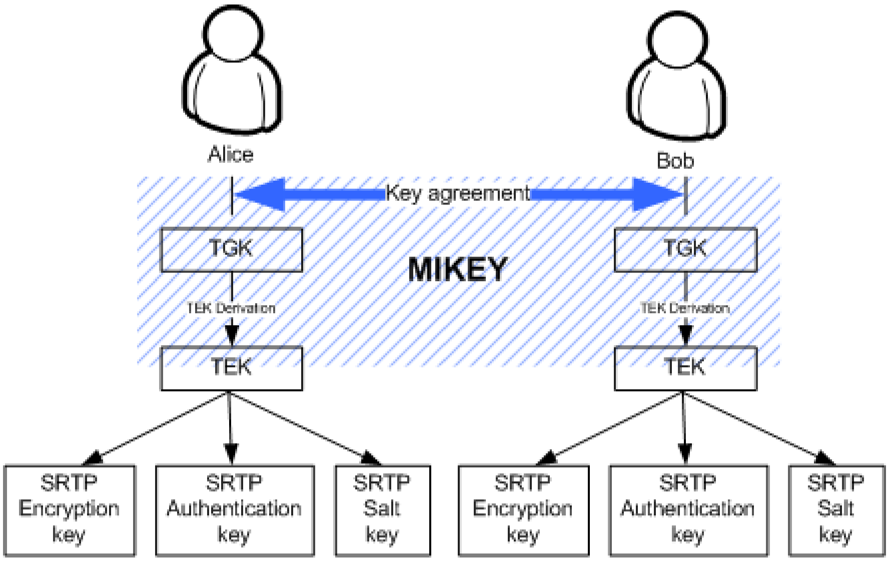


**Figure 18** MIKEY operation

#### ***Solved security issues***

Several security issues can be protected by using MIKEY. The most important ones are the following:

- **Mutual authentication:** it allows the authentication of both parts in the media session by the use of a challenge/response scheme.

  To fit the challenge/response scheme in the offer/answer model, basically reducing by one the number of exchanged messages, MIKEY uses timestamps as challenges.

  Using timestamps in combination with an authentication key, allows the participants to verify the identity of each other.

- **Replay protection:** it prevents the use of the same timestamp to authenticate for twice.

  When a timestamp is received, it is stored locally and every new MIKEY message should use a later timestamp than the stored ones.

- **Perfect forward secrecy:** using Diffie-Hellman agreement schema, it is possible to assure that even if a private key is revealed, there is no chance to compromise the keys used in a crypto session.

  This is possible because a completely new and random TGK is agreed every time a connection starts.

### *Key agreement types*

Depending on the computational resources and the authentication procedures available, MIKEY describes three types of key agreements [20]:

- **Pre-shared key (PSK):** in this type of agreement, Alice and Bob must share a secret in advance.

  Assuming K as the pre-shared secret, RAND as random generated string, ID as the user identifier, constant1 and constant2 as two well-known byte strings and f() as a pseudo-random function defined in the MIKEY draft, the key agreement would work as shown in Figure 19.

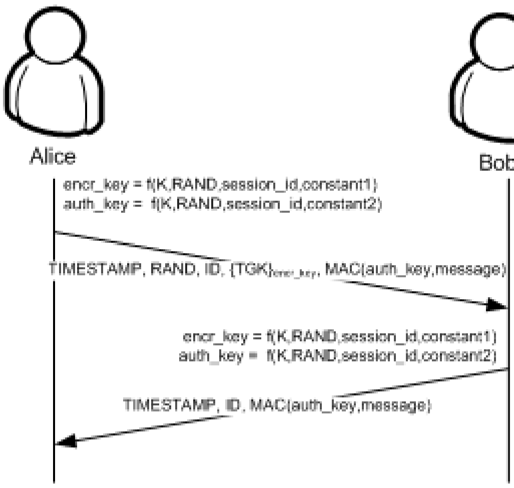

Alice
Bob
encr_key = f(K,RAND,session_id,constant1)
auth_key = f(K,RAND,session_id,constant2)
TIMESTAMP, RAND, ID, {TGK}encr_key, MAC(auth_key,message)
encr_key = f(K,RAND,session_id,constant1)
auth_key = f(K,RAND,session_id,constant2)
TIMESTAMP, ID, MAC(auth_key,message)

**Figure 19** MIKEY PSK key agreement

Alice generates an encryption key and an authentication key. Then she sends a Timestamp, a random string, her user identifier, the MIKEY TGK key encrypted with the generated encryption key and a MAC to protect the whole message using the generated authentication key.

Then, Bob generates the encryption key and the authentication key with the parameters he received from Alice. These keys should be the same as Alice generated, since the parameters used to generate them are the same. So then, Bob can decrypt the TGK key and can check the MAC Alice sent.

After that, Bob answers with a Timestamp, his user identifier and a MAC to protect the message with the generated authentication key.

Finally, Alice checks the MAC to make sure that Bob is the one who sent this message.

- **Public-key encryption (PKE):** in this type of agreement, Alice and Bob must have each a pair of public/private keys. It is also needed a PKI in order to validate user keys in front of other users.

  Assuming rand() as a random function that generates a random byte stream, RAND as random generated string, ID as the user identifier, constant1 and constant2 as two well-known byte strings and f() as a pseudo-random function defined in the MIKEY draft, the key agreement would work as shown in Figure 20.

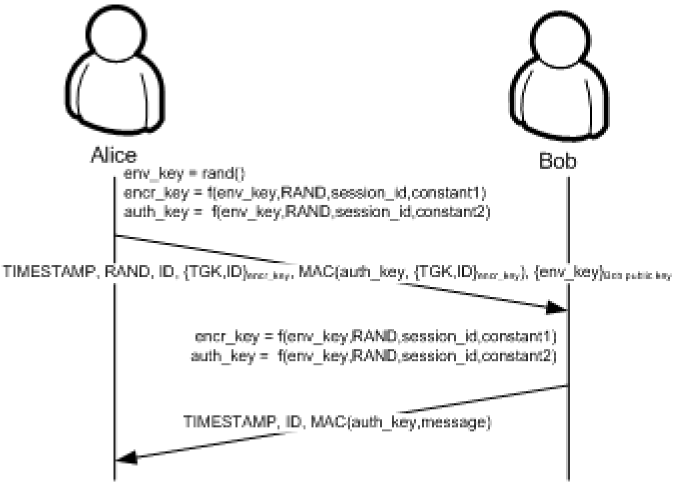

Alice
Bob
env_key = rand()
encr_key = f(env_key,RAND,session_id,constant1)
auth_key = f(env_key,RAND,session_id,constant2)
TIMESTAMP, RAND, ID, {TGK,ID}encr_key, MAC(auth_key, {TGK,ID}encr_key), {env_key}Bob public key
encr_key = f(env_key,RAND,session_id,constant1)
auth_key = f(env_key,RAND,session_id,constant2)
TIMESTAMP, ID, MAC(auth_key,message)

**Figure 20** MIKEY PKE key agreement

Alice generates an envelope key with a random function. With the result of that function, an encryption key and an authentication key are generated. Then she sends a Timestamp, a random string, her user identifier, the MIKEY TGK key encrypted with the generated encryption key, a MAC to protect the whole message using the generated authentication key and the envelope key encrypted with Bob's public key.

Then, Bob decrypts the envelope key using his private key. With the clear envelope key, he generates the encryption key and the authentication key with the other parameters he received from Alice. These keys should be the same as Alice generated, since the parameters used to generate them are the same. So then, Bob can decrypt the TGK key and can check the MAC Alice sent.
After that, Bob answers with a Timestamp, his user identifier and a MAC to protect the message with the generated authentication key.

Finally, Alice checks the MAC to make sure that Bob is the one who sent this message.

- **Signed Diffie-Hellman (DH):** in this type of agreement, Alice and Bob must have each a pair of public/private keys. It is also needed a PKI in order to validate user keys in front of other users and complicate man-in-the-middle attacks, since the attacker would need both Alice an Bob private keys.

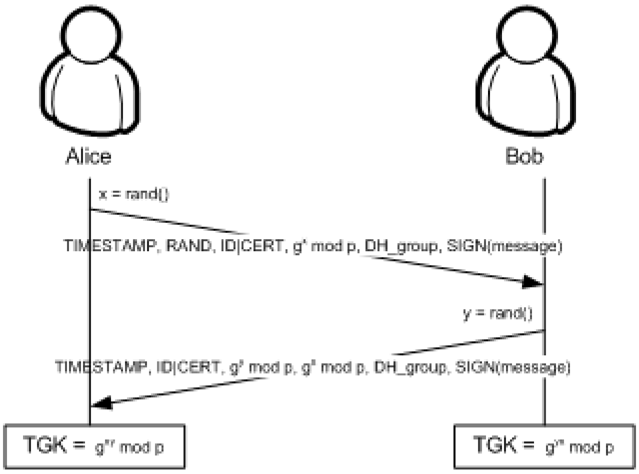

Alice
Bob
x = rand()
TIMESTAMP, RAND, ID|CERT, $g^x$ mod p, DH_group, SIGN(message)
y = rand()
TIMESTAMP, ID|CERT, $g^y$ mod p, $g^x$ mod p, DH_group, SIGN(message)
TGK = $g^{xy}$ mod p
TGK = $g^{yx}$ mod p

**Figure 21** MIKEY Signed DH key agreement

Alice generates a random number (x) and sends to Bob a Timestamp, a random byte stream, her user identifier or her certificate, $g^x$ mod p, a set of parameters used in the Diffie-Hellman algorithm (basically g and p, defined by a DH_group identifier) and a signature of the message.

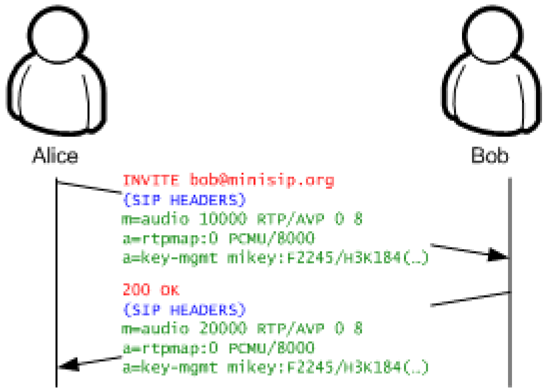
Bob generates a random number (y) and answers Alice with a Timestamp, his user identifier or his certificate, $g^y$ mod p, $g^x$ mod p, the DH_group and a signature of the whole message.

Then, both Alice and Bob can calculate a TGK as $g^{xy}$ mod p. This TGK is generated randomly, so that is why is said that Signed Diffie-Hellman key exchange preserves the perfect forward secrecy, since a new key agreement would generate a completely different TGK [13] [20].

### ***MIKEY in the SIP dialog***

One of the main design goals of MIKEY is to fit the key agreement in the media negotiation process, usually done by SIP. The reason why MIKEY should perform the key agreement and the mutual authentication in just one roundtrip is that the SIP call establishment is done in just one roundtrip.

To achieve that, MIKEY key agreement is performed in the INVITE transaction, by adding a new SDP attribute: `key-mgmt`. This attribute can be set as a session attribute or as a media attribute (see Figure 22) [14].
In the first case, the exchanged TEK will protect all the streams agreed in that session. In the second, a TEK will be exchanged for every flow described and it will be used only to protect only media in that flow.



```
INVITE bob@minisip.org
(SIP HEADERS)
m=audio 10000 RTP/AVP 0 8
a=rtpmap:0 PCMU/8000
a=key-mgmt mikey:F2245/H3K184(..)
```

```
200 OK
(SIP HEADERS)
m=audio 20000 RTP/AVP 0 8
a=rtpmap:0 PCMU/8000
a=key-mgmt mikey:F2245/H3K184(..)
```

**Figure 22** SIP and MIKEY

In case re-keying is needed, it should be performed by a `RE-INVITE` transaction, conducted the same way as the `INVITE` transaction did.

Dealing with non-secured SIP UAs should also be considered. If a secured UA invites a non-secured UA, the latter is going to omit the `key-mgmt` attribute in the SDP of the `200 OK` message. In that case, the first UA should give the user the option to fall back to an unsecured session initiation (adding another SDP media descriptor in the `INVITE` message) or to cancel the communication.

In the last case, when the callee UA does not accept a non-secure call and the caller policy is do not accept unsecured calls, the receiver could suffer from ghost ringing. Since the `200 OK` message is not sent until the callee accepts the call, he doesn't know if Alice will reject the non-secure session and close the establishment.

The solution to this problem is achieved by using the `1XX` SIP responses, provisional responses, to carry the MIKEY response. If the `180 Ringing` is used to carry the MIKEY response, the ringing delay could be big, since the callee should process all the crypto session.  The solution for that is to use a `183 Session in Progress` message, sent after the `180 Ringing` but before the `200 OK` [16].

# 3. Background study conclusions

After looking at the state of the art of several technologies linked to the area of this study, several conclusions in the different subjects covered can be proposed.

## 3.1. Video conferencing

Looking at the current video conference solutions, their problems and some solutions proposed in other studies and the lack of multicast capabilities of most networks, I conclude that a Multi-point Conference Unit is nowadays the best option for video conferencing over Internet.

This system allows the users to send their media flows just once, but these are finally delivered to all other participants in the video conference.

However, most of the commercial solutions are based in proprietary systems integrated in dedicated devices. Although most of them use standard protocols (such as H.323 and SIP), solve some security issues and can deal with High Definition media, they are closed solutions.

Also, it is necessary to work in the video CODEC issue in order to minimize the delay introduced in the compression process.

Finally, the need of a High Definition video conference system comes up to give the user the feeling he or she is in a real meeting with other participants. Adding other capabilities such as sharing files or digital shared blackboards could really make the difference in front of traditional video conference systems.

## 3.2. Signalling protocol

In this study I dealt with the signalling of a media session establishment, looking at the two main protocols used for that, H.323 and SIP.

After my analysis, I come to the conclusion that because of its flexibility and its simplicity, SIP is the right protocol to use in the development of a new video conference environment.

Moreover, SIP offers several options to secure the signalling messages and to perform user authentication, even though it is done using other protocols, and the possibility to establish a secure media session using SRTP and MIKEY.

## 3.3. Security

Several studies [18] show that using SRTP is the best option to secure a media session. This option does not add too much overhead and the security standards used on it are quite fast. Also, security is added at the application layer and can be platform-cross, since other options such as IPSec are usually linked to the operating system.

These reasons make me think that SRTP should be a very good solution for video conferencing, where minimizing the delays is one of the most important issues to solve. Also, thinking about High Definition or high quality media flows and their high bit rate, reducing the overhead is also important in order to save network bandwidth.

SRTP needs an external mechanism to perform the key exchange, and previous studies at TSLab were focused on the use of MIKEY to do so.

Since MIKEY is thought to be used over SIP and I stated that SIP is the best option for video conference signalling, I see no reason not to use also MIKEY as the exchange mechanism for SRTP master keys.

# 4. Implementation proposal

After a theoretical study over different technologies that could be involved in a video conference solution and after stating which of them would fit better in this environment, I made a proposal on what I would like to implement.

My idea consisted in building a secure MCU based on the i2cat RTP Packet Reflector and using miniSIP libraries to establish a secure media session.

The RTP Packet reflector is a tool that receives RTP flows from all participants in a session and forwards one or all of these flows to all the participants. At that time, i2cat HDVIPER team was still working on some improvements of this software, in order to make it more flexible.

The operation of this Packet Reflector is based on the decisions made by a meeting director, who would be able to switch the flows to be shown to participants in a meeting. My proposal was to create a mechanism to perform this actions using SIP.

miniSIP is not only a SIP User Agent but a set of libraries that can be used to build new SIP applications. miniSIP also implements some of securing SIP measures, the MIKEY key agreement and can deal with SRTP media flows.

Taking into account both software pieces, I proposed to create a SIP layer (using miniSIP libraries) to deal with all the signalling involved in the video conference set up, user joining and flow switching. This SIP layer should be able to communicate with the RTP Packet Reflector through a well-defined API (Figure 23), which was agreed with i2cat team, in order to make changes at the media level (switch the showed flow/s).

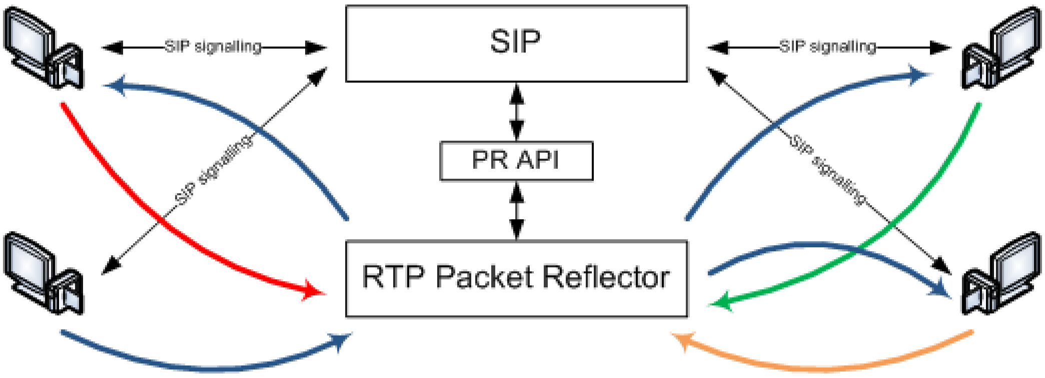


**Figure 23** MCU layer proposed design

After finishing the signalling layer and its integration with the Packet Reflector, some changes were done in the RTP Packet reflector in order to add the possibility to work with SRTP flows (Figure 24). This drove me to add a MIKEY agreement using the miniSIP libraries and to add the parameters needed for it in the Packet Reflector.

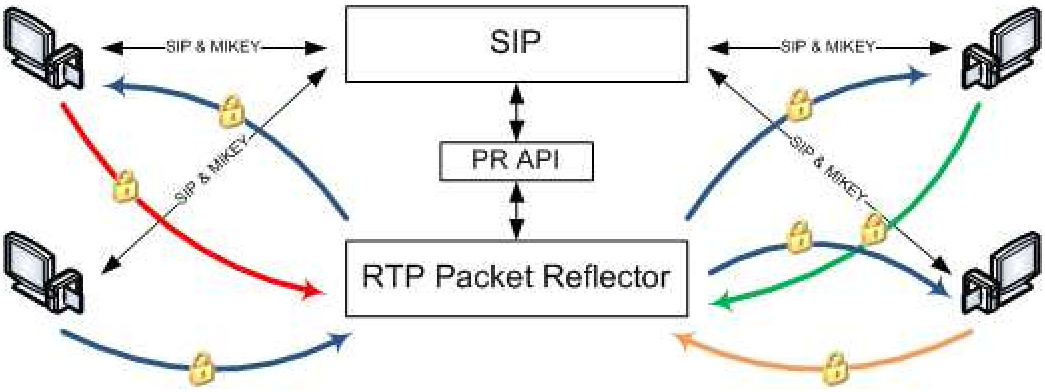


**Figure 24** SRTP capable MCU

In order to control the flow switching by the director of the meeting, a signalling mechanism should be established (Figure 25). I proposed to use SIP, probably adding a new type of message to carry the flow switching commands. I finally discarded this feature, since the MCU implemented a forwarding-all-to-all behaviour (see 5, Implementation).

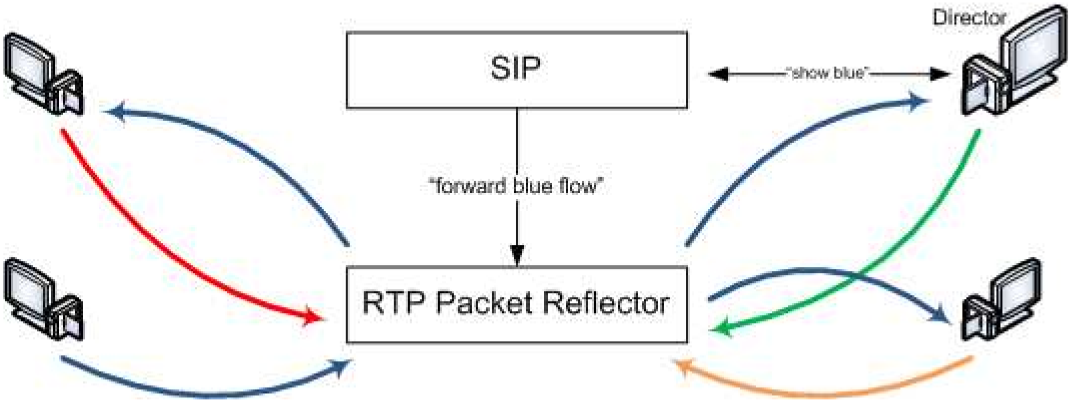


**Figure 25** Director Functionality

In terms of measurements, I would like to measure the delay introduced in the Packet Reflector by forwarding the RTP packets as they come from their sources. This could be done using the RTP Packet reflector and some time measurement tools.

After adding the SRTP functionality, I repeated the same measurements done in the RTP Packet Reflector with the secured software. Then I compared the results with the ones obtained in the non-secured software.

Finally, I analyzed the obtained result in the secured scenario and determined if it was possible to use SRTP in a MCU based video conference, Trying to figure out how many participants can be handled by an MCU and which was the maximum flow bit rate they can send would be interesting too.

# 5. Implementation

As stated in the implementation proposal (see 4), me and Erik Eliasson (my advisor) used miniSIP libraries to build a SIP layer for the Packet reflector, in order to create a media flow forwarding MCU. After that, we used one of the miniSIP libraries (*libmikey*) to perform a key exchange and set up SRTP sessions, securing the flows forwarded by the Packet Reflector.

All the implementation was done using C++ language and tested under Ubuntu Linux 7.04. Since the libraries used to build the software are platform-cross, the result should be also platform-cross, although it hasn't been tested in other platforms than Linux.

## *5.1. SIP layer*

This element is the responsible of the establishment and control of the video conference through the MCU. It is SIP compliant, so any SIP client should be able to join a conference using this software.

It works as a SIP server, receiving calls from the participants and adding them to the conference they ask to get in. It is also capable to perform a key exchange using the MIKEY protocol. The keys exchanged are used in the Packet Reflector to secure the media flows using SRTP.

It has been created using miniSIP libraries, a tool developed mainly in KTH and as an OpenSource project. Specifically, *libmutil*, *libmnetutil*, *libmcrypto*, *libmikey*, *libmsip* and *libminisip* were used in the implementation of this application.

### 5.1.1. Conferences

First of all, the administrator should set up a conference in the service. At the moment, this can be done using the command line to create rooms.

Once the room is created, the admin can decide which SIP-URIs are allowed to be in that conference. There are several possibilities to set up access policies:

- Allow all SIP-URIs, using `*`.
- Allow SIP-URIs just from one domain, using `*@domain.com`.
- Allow only specific SIP-URIs, using `user@domain.com`.

The command line interface accepts the next commands to manage the conferences running in the service:

- **`room add [name] [description]` :** adds a new conference, identified by its name
- **`room del [name]` :** deletes a conference
- **`room allow [room_name] [SIP-URI]` :** allow SIP-URI(s) in the conference
- **`room deny [room_name] [SIP-URI]` :** remove allowed SIP-URI(s) in the conference

- **`room list :`** lists the active conferences in the service
- **`room list allowed [room_name] :`** lists the allowed SIP-URIs in the conference

Finally, a default room is created by default and anybody can get in. This is done in order to allow SIP clients that don't support tags in the URI to get in a conference.

### 5.1.2. SIP calls

As soon as the conferences are set up in the service, any of the SIP-URIs allowed to get in the conferences could call to the service. To do so, a room tag should be specified in the SIP-URI called. An example for that could be `sip:mcu@minisip.org`**`;room=testing`**.

The service will check if the incoming call comes from a SIP-URI allowed to be in the room specified. If so, the call will be accepted in the required conference, and in case not, the call will be placed in the default room.

### 5.1.3. SDP media dealing

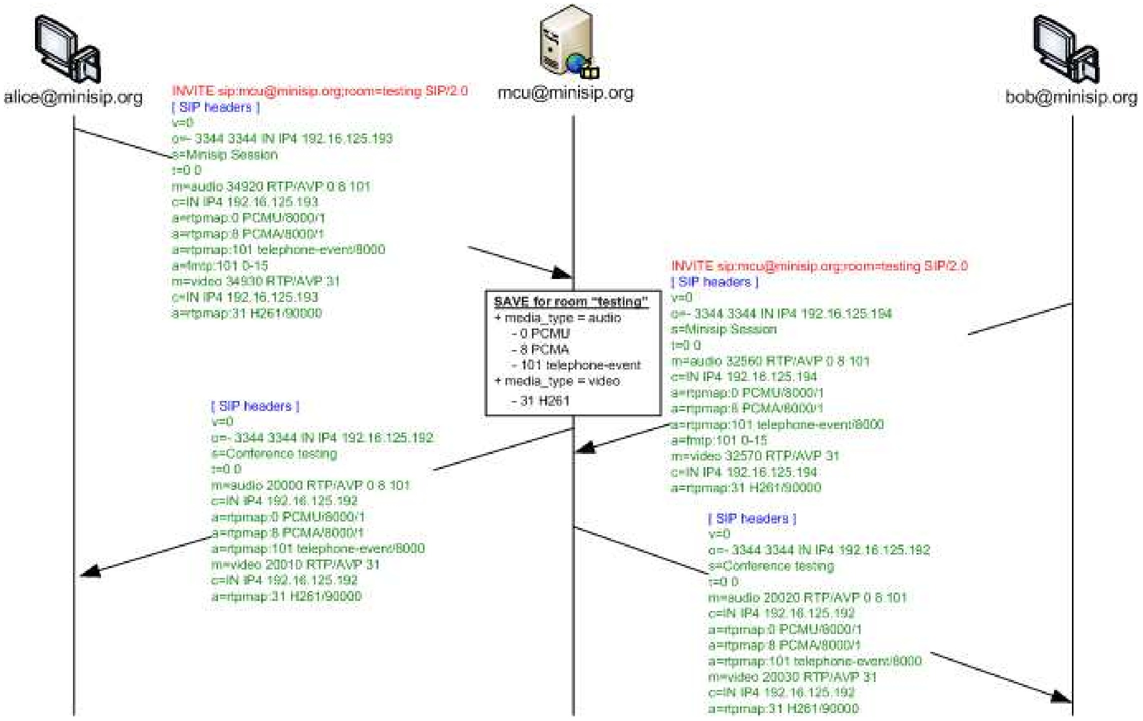
Since the MCU is going to deal with several SIP clients that will not be calling each other directly, some measures should be taken in order to make them agree which media to use. The main problem here is that the SIP client is not negotiating the media session with the final destination of the media, so some problems (such as incompatible types of media, media identifiers in RTP not in common) showed up.

To solve this, when a call arrives to the MCU, the SDP offer carried in the SIP `INVITE` is analyzed by the MCU.

If the user is the first one to get into he conference, all the rtpmap parameters on the SDP are stored (specially the RTP media type). When other users get in the conference, they must support at least all the media stored by the MCU and all the RTP media types should be the same.

Operating this way is a simple solution to make sure everybody is available to deal with media that other participants will send, but also had a very important problem: only clients supporting the media supported by the first client to join the conference would be able to get in.

An example for that would be the following:



```
INVITE sip:mcu@minisip.org;room=testing SIP/2.0
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.193
s=Minisip Session
t=0 0
m=audio 34920 RTP/AVP 0 8 101
c=IN IP4 192.16.125.193
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 34930 RTP/AVP 31
c=IN IP4 192.16.125.193
a=rtpmap:31 H261/90000
```

```
SAVE for room "testing"
+ media_type = audio
   - 0 PCMU
   - 8 PCMA
   - 101 telephone-event
+ media_type = video
   - 31 H261
```

```
INVITE sip:mcu@minisip.org;room=testing SIP/2.0
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.194
s=Minisip Session
t=0 0
m=audio 32560 RTP/AVP 0 8 101
c=IN IP4 192.16.125.194
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 32570 RTP/AVP 31
c=IN IP4 192.16.125.194
a=rtpmap:31 H261/90000
```

```
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Conference testing
t=0 0
m=audio 20000 RTP/AVP 0 8 101
c=IN IP4 192.16.125.192
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
m=video 20010 RTP/AVP 31
c=IN IP4 192.16.125.192
a=rtpmap:31 H261/90000
```

```
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Conference testing
t=0 0
m=audio 20020 RTP/AVP 0 8 101
c=IN IP4 192.16.125.192
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
m=video 20030 RTP/AVP 31
c=IN IP4 192.16.125.192
a=rtpmap:31 H261/90000
```

**Figure 26** SDP media dealing

We chose this implementation for its simplicity, but it is one of the features that must be improved in future versions of the software.

### 5.1.4. SIP State machine

The implementation of the SIP layer performs the following state machine:

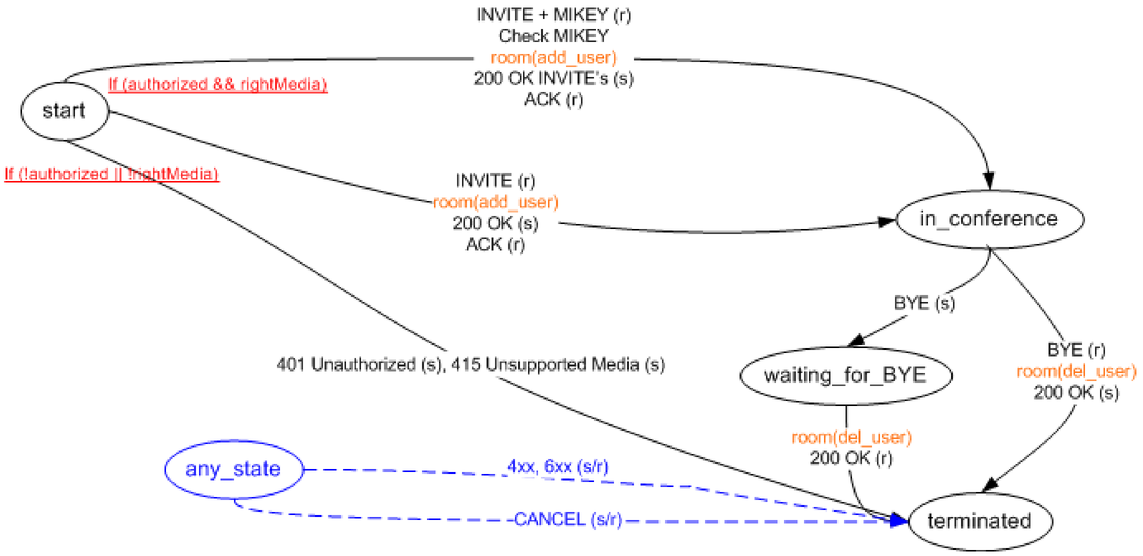


**Figure 27** Implemented SIP state machine

When an incoming call arrives, the MCU checks if the user is authorized to be in the conference he is asking to participate in (see 5.1.1 Conferences). It also checks if the user is ready to deal with all types of media in the conference. If one of these conditions is not fulfilled, the MCU rejects the call by a 4xx SIP message.

If the `INVITE` message carries a MIKEY message, the MCU computes the response and attaches it to the `200 OK` response. Then the user is added to the conference he asked to participate in (or to the default one) and the MCU goes to the state *in_conference*.

If the *INVITE* message is not carrying a MIKEY message, the MCU just adds the user to the conference and sends the `200 OK` response. Finally, it goes to the *in_conference* state.

When a user leaves the conference, the MCU receives the `BYE` message, responds with a `200 OK` and deletes the user from the conference. Finally, the call is terminated as going to the *terminated* state. However, the MCU can also terminate one call: it sends a `200 OK`, goes to the *wait_for_BYE* state, waits for the `200 OK`, remove the user from the conference and go to the *terminated* state.

### 5.1.5. Key exchange: MIKEY

In order to use SRTP to protect the media, each client has to exchange a key with the MCU, using the MIKEY protocol. Therefore, if the user requires the conference to be secure, a MIKEY capable SIP client must be used.

In this implementation, only the Pre-shared Key method is implemented. It is quite simple, but the security level is enough to protect media flows in a video conference environment.

During the implementation, we solved some issues related to the parameters used to derive keys appeared during the process. As stated in the background study of the MIKEY protocol (2.4.4 Key Agreement: MIKEY), MIKEY exchanges a Transport Generation Key (TGK). From this key, several Transport Encryption Keys (TEKs) could be derived, using the Crypto Session identifier (sent in the MIKEY exchange process) as the deriving index (see Figure 18).

Again, it is possible to derive several sets of keys (encryption key, authentication key and salt key) to be used in the SRTP from a TEK, using the SSRC as the deriving index (see Figure 18).

For every RTP flow to be secured, a crypto session with its identifier is going to be created and exchanged in MIKEY. The problem appeared when the MCU sent a flow with an SSRC that was not signalled in the MIKEY key exchange process, since the destination is not aware of that SSRC.

Our solution was to wait for the first RTP packet of each SSRC to generate the SRTP keys. We read the SSRC from this first packet (the RTP headers are not encrypted) and then the MCU is able to derive the keys to use in that flow from the TEK generated for that call.

The same happened when forwarding a new flow to one of the participants. Since the SSRC was not announced in the MIKEY exchange, the participant does not know about it. The MCU will use the TEK generated for this call and is going to generate the SRTP keys using the newly appeared SSRC.

Finally, the client will read the SSRC in the first packet of the SRTP flow, will generate the new keys and will decrypt and authenticate the flow. We also needed to change that in the miniSIP client.

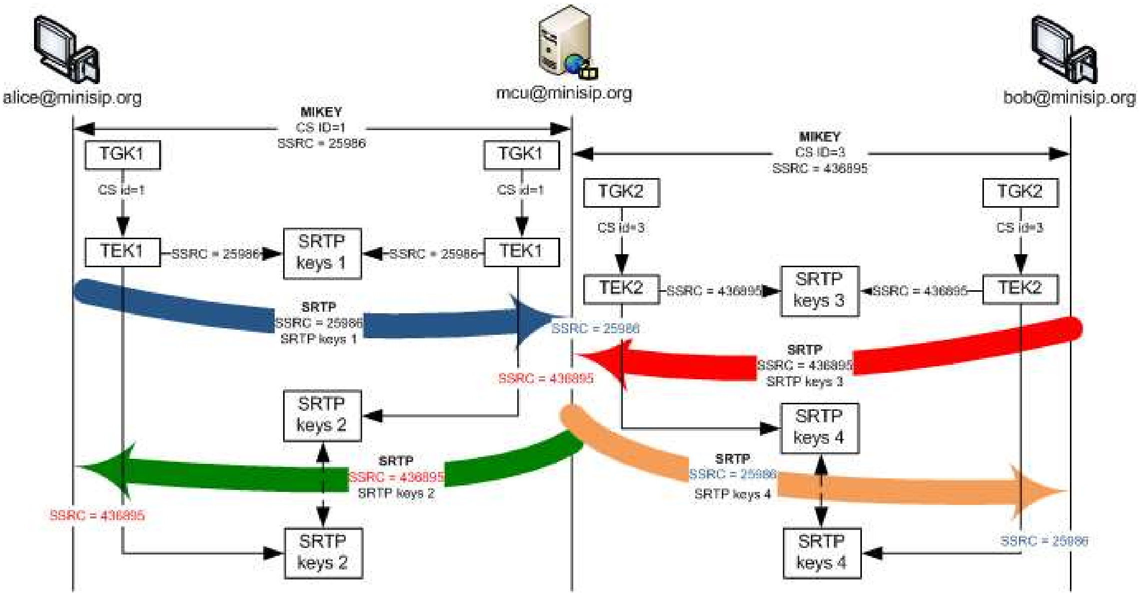


**Figure 28** Dynamic SRTP key generation

### *Re-Keying feature*

In the theoretical study of the SRTP is stated that a re-keying mechanism is needed if more than $2^{48}$ packets are encrypted with the same key. Although MIKEY is capable to do that, we realised this feature is not needed in our MCU.

We made some calculations in different cases to demonstrate that the re-keying feature is not needed. We calculate how much time would take to send $2^{48}$ packets with some well-known bandwidth rates of some media generators.

- **PCMU ~ 64kbps (8 Kbytes/s)**

$$t_{packet} = \frac{160 bytes / packet}{8 \cdot 10^{3} bytes / s} = 20ms$$

$$T_{2^{48} packets} = t_{packet} \cdot 2^{48} packets = 20 \cdot 10^{-3} \cdot 2^{48} \approx 178510 years$$

- **HDV over IP (DVTS) ~ 25Mbps (3,125Mbytes/s)**

$$t_{packet} = \frac{1480 bytes / packet}{3{,}125 \cdot 10^{6} bytes / s} = 473{,}6 \mu s$$

$$T_{2^{48} packets} = t_{packet} \cdot 2^{48} packets = 473{,}6 \cdot 10^{-6} \cdot 2^{48} \approx 4227 years$$

- **SMTPE 292M over IP (UltraGrid) ~ 1,5Gbps (187,5Mbytes/s)**

$$t_{packet} = \frac{1480 bytes / packet}{187,5 \cdot 10^{6} bytes / s} = 7,893 \mu s$$

$$T_{2^{48} packets} = t_{packet} \cdot 2^{48} packets = 7,893 \cdot 10^{-6} \cdot 2^{48} \approx 70 years$$

Even if we consider using the same key for the flow sent and the flow received (the time would be the half), we realise that the re-keying feature is not needed at all. Moreover, a possible attacker should sniff all the traffic during 35 years in the worst case and then analyse this traffic in order to calculate the keys used to secure that information.

## *5.2. Packet Reflector*

This piece of software was developed by the i2cat HDVIPER team. It is an RTP packet bouncer that is capable to forward several RTP media flow, each one to several destinations.

This software is also written in C++, so it was easy to integrate in the new MCU application created with miniSIP libraries.

### 5.2.1. Integration with the SIP Layer

The packet reflector is compiled in the same binary file as the SIP Layer does, so the result of the application is just one executable file.

In order to build a complete MCU, it is needed not only the establishment but also to deal with media. In this case, we are building a forwarding MCU, so we need to tell the Packet Reflector what and where to forward it.

To do so, an API was defined with the i2cat team. With this API, we should be able to add or delete sources and destinations in the Packet Reflector. These sources and destinations are agreed between participants and the MCU using SIP and SDP.

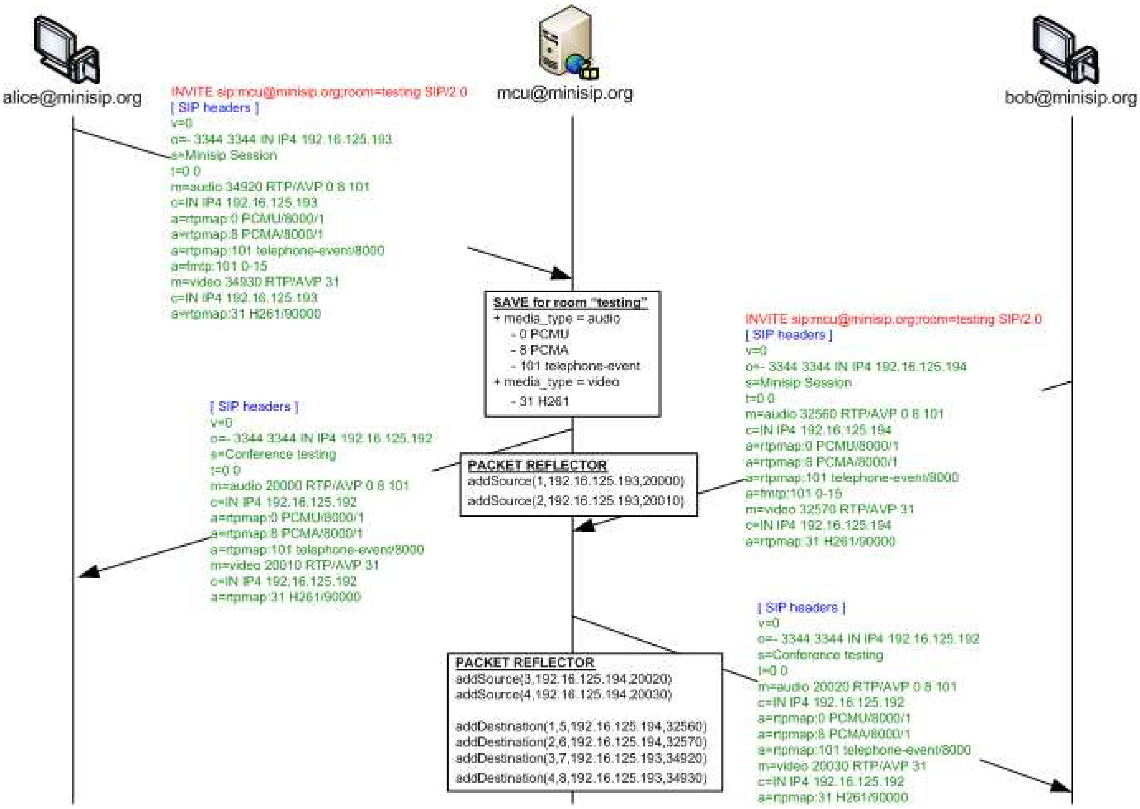
When the MCU receives a valid incoming call (that is allowed and supports the required media), the information related to the media flows (the type of media, the IP address and the port) is read and an SDP answer is sent. After that and taking into account the parameters agreed in the SDP negotiation, the flows are set up in the Packet Reflector.

The basic API for the Packet Reflector has the following methods and parameters:

- `addSource(flowId, source IP, destination Port)`: adds a media flow as a source
- `addDestination(sourceFlowId, flowId, destination IP, destination Port)`: adds a destination where to forward one of the source flows
- `removeSource(flowId)`: removes one of the created sources and all the destinations for it

- `removeDestination(sourceFlowId, flowId)` : removes a destination of one of the sources

Using the same example used in SDP media dealing, the Packet Reflector would receive the next orders:



```
INVITE sip:mcu@minisip.org;room=testing SIP/2.0
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.193
s=Minisip Session
t=0 0
m=audio 34920 RTP/AVP 0 8 101
c=IN IP4 192.16.125.193
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 34930 RTP/AVP 31
c=IN IP4 192.16.125.193
a=rtpmap:31 H261/90000
```

```
SAVE for room "testing"
+ media_type = audio
   - 0 PCMU
   - 8 PCMA
   - 101 telephone-event
+ media_type = video
   - 31 H261
```

```
PACKET REFLECTOR
addSource(1,192.16.125.193,20000)
addSource(2,192.16.125.193,20010)
```

```
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Conference testing
t=0 0
m=audio 20000 RTP/AVP 0 8 101
c=IN IP4 192.16.125.192
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
m=video 20010 RTP/AVP 31
c=IN IP4 192.16.125.192
a=rtpmap:31 H261/90000
```

```
INVITE sip:mcu@minisip.org;room=testing SIP/2.0
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.194
s=Minisip Session
t=0 0
m=audio 32560 RTP/AVP 0 8 101
c=IN IP4 192.16.125.194
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 32570 RTP/AVP 31
c=IN IP4 192.16.125.194
a=rtpmap:31 H261/90000
```

```
PACKET REFLECTOR
addSource(3,192.16.125.194,20020)
addSource(4,192.16.125.194,20030)

addDestination(1,5,192.16.125.194,32560)
addDestination(2,6,192.16.125.194,32570)
addDestination(3,7,192.16.125.193,34920)
addDestination(4,8,192.16.125.193,34930)
```

```
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Conference testing
t=0 0
m=audio 20020 RTP/AVP 0 8 101
c=IN IP4 192.16.125.192
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
m=video 20030 RTP/AVP 31
c=IN IP4 192.16.125.192
a=rtpmap:31 H261/90000
```

**Figure 29 Packet** Reflector orders in a SDP negotiation

After that, the MCU will forward the media flows as shown in Figure 30.

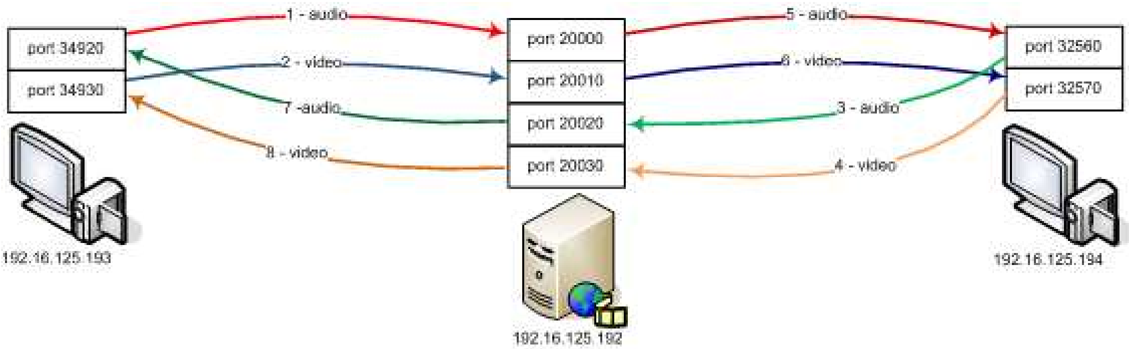


**Figure 30** Flow result

The `addSource` and `addDestination` methods are called when a user joins the conference (see `room(add_user)` in Figure 27). The SIP Layer reads the SDP information and sets up the flows in the Packet Reflector using that information.

Finally, the `removeSource` and `removeDestination` methods are called when a user leaves the conference (see `room(del_user)` in Figure 27). The

SIP Layer keeps track of the flows assigned to a certain user, so when it leaves, these flows are deleted in the Packet Reflector.

### 5.2.2. Securing media flows: SRTP

In order to add security to the media flows forwarded by the MCU, we will use the SRTP protocol.

Since the users are establishing the session with the MCU but the real destination of their flows is not the MCU, we need it to decrypt and re-encrypt all the secured media flows.

The keys will be generated as explained in 5.1.5 (Key exchange: MIKEY) and the operation will work as shown in Figure 31.

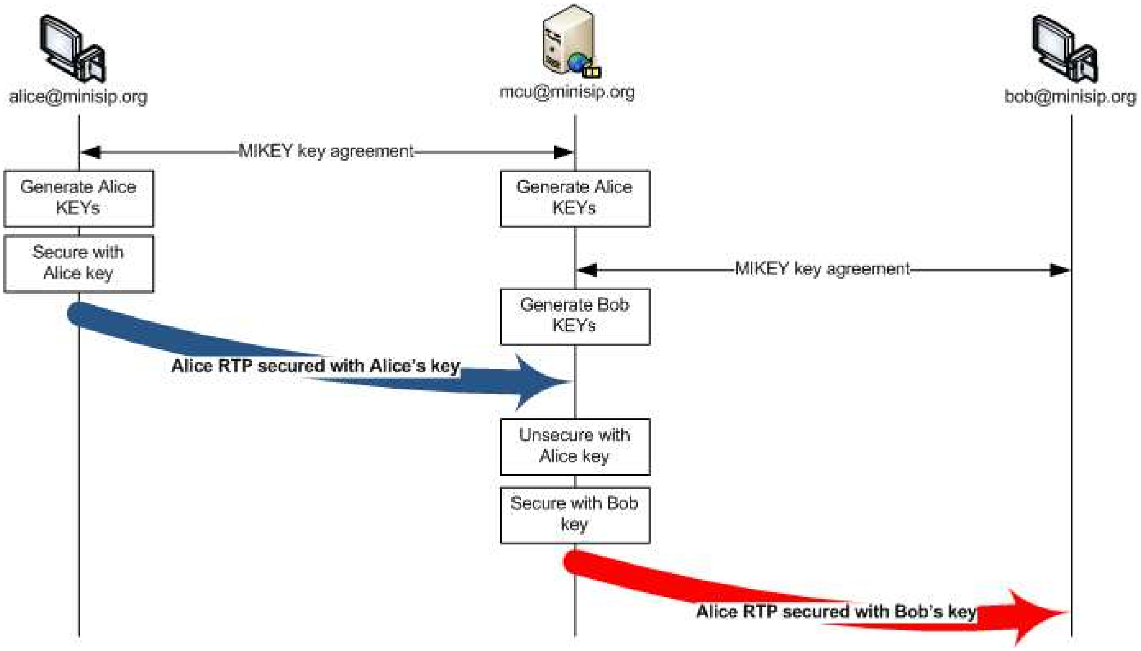


**Figure 31** MCU operating with SRTP flows

In order to generate the right keys for each user, we added new methods to create secured flows in the Packet Reflector. These methods just added a link with the MIKEY information exchanged with the user, so the right TEK can be used to generate the SRTP keys. The new methods added are the following ones:

- `addSource(flowId, source IP, destination Port, MIKEY info)` : adds a media flow as a source, telling the Packet Reflector that all packets from this source should be decrypted and authenticated using the given MIKEY information
- `addDestination(sourceFlowId, flowId, destination IP, destination Port, MIKEY info)` : adds a destination where to forward one of the source flows, telling the Packet Reflector that all packets sent to this destination should be encrypted and authenticated using the given MIKEY information

## 5.3. *Alternative implementation*

We had several other ideas to solve the problems shown up during the design and implementation process. Although we did not implement them, they should be taken into consideration for future improvements on the software.

### 5.3.1. Calling process

At the beginning of the design process, we considered two options for the MCU operation:

- Make the MCU to act as a server, so users could call to get into a conference.

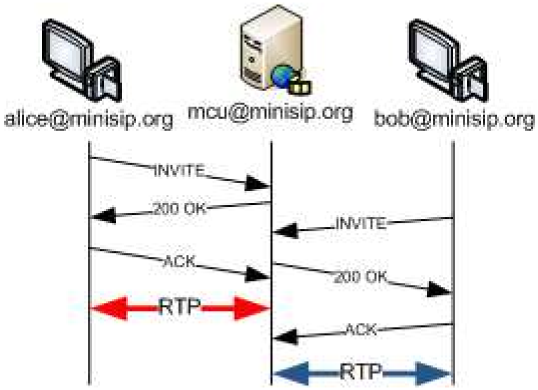


**Figure 32** MCU as a server

- Create a mechanism to make the MCU to call the participants in a conference, so they just have to answer the call to be in a conference scheduled at a certain time.

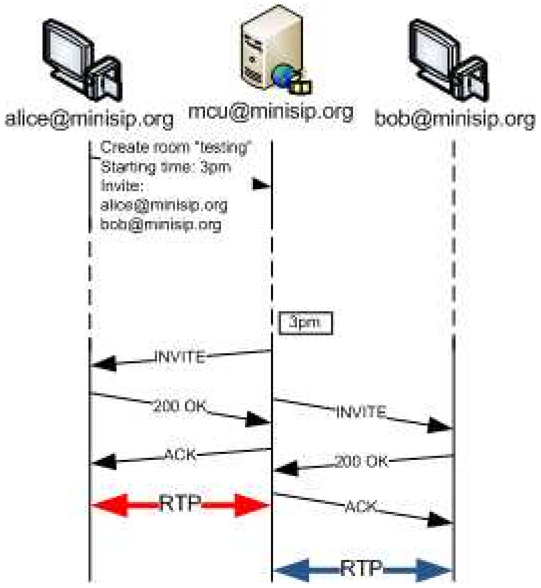


**Figure 33** MCU as a caller

Both ideas were right, but we preferred the first one, since we wish the user to require being in a conference. Moreover, in the second option we would have to build a mechanism to schedule the MCU to call to certain users at a certain time, although it could also have been good for resource planning purposes.

However, would be nice to mix both in future versions. This would allow the system to invite new participants during a conference.

### 5.3.2. Configuring rooms

In this first implementation, we chose a command line system to add and configure conferences in the MCU, as explained in 5.1.1 (Conferences). We implemented this because it was very simple and we did not need something more complex at that time.

A very good improvement for this system would be to create a web interface in order to allow the administrator to manage conferences. From this interface, it would be possible to add or delete rooms, allow or deny users and define schedules for the conference, so the resources could be booked. The best advantage of this option is that just a browser would be needed to manage the MCU.

We did not look much into this feature, but we thought that a good system would be to use a database system to store data and a small daemon reading it, as done in ComCloser [4]. For this purpose, SQLite [25] seems to fit perfectly, since only a single file is needed and there is a small driver implemented in C++ that offers the possibility to push an alarm when content is modified.

### 5.3.3. Signalling the conference

In order to signal to the MCU to which conference the user wants to get in, our first thought was to use a different SIP URI for each room.

Our idea was to make the MCU to register as a new SIP URI against the Registrar for each room created in the system. Then, the users just had to call to a simple SIP URI, such as `room_name@domain.com`.

We discarded this idea because it was needed for the MCU to register the same contact info for several SIP URIs. We believed that using SIP parameters in the SIP URI was a better solution for this problem.

### 5.3.4. Media parameters negotiation

In 5.1.3 (SDP media dealing), it is explained the problem appeared when dealing with the media session negotiation in a conference using an MCU.

We thought about different possibilities, a part of the chosen one, to solve this issue:

- Check just the media name and save the media type identifier. After that, modify the forwarded packets with the media type negotiated with the destination. We discarded this idea because it was not the most efficient one since media packet should be analyzed and modified on the fly.
- Find the common CODECs between the MCU and a new participant. If there is any common CODEC, accept the call but delete from the accepted media list the CODECs that this new user does not support.

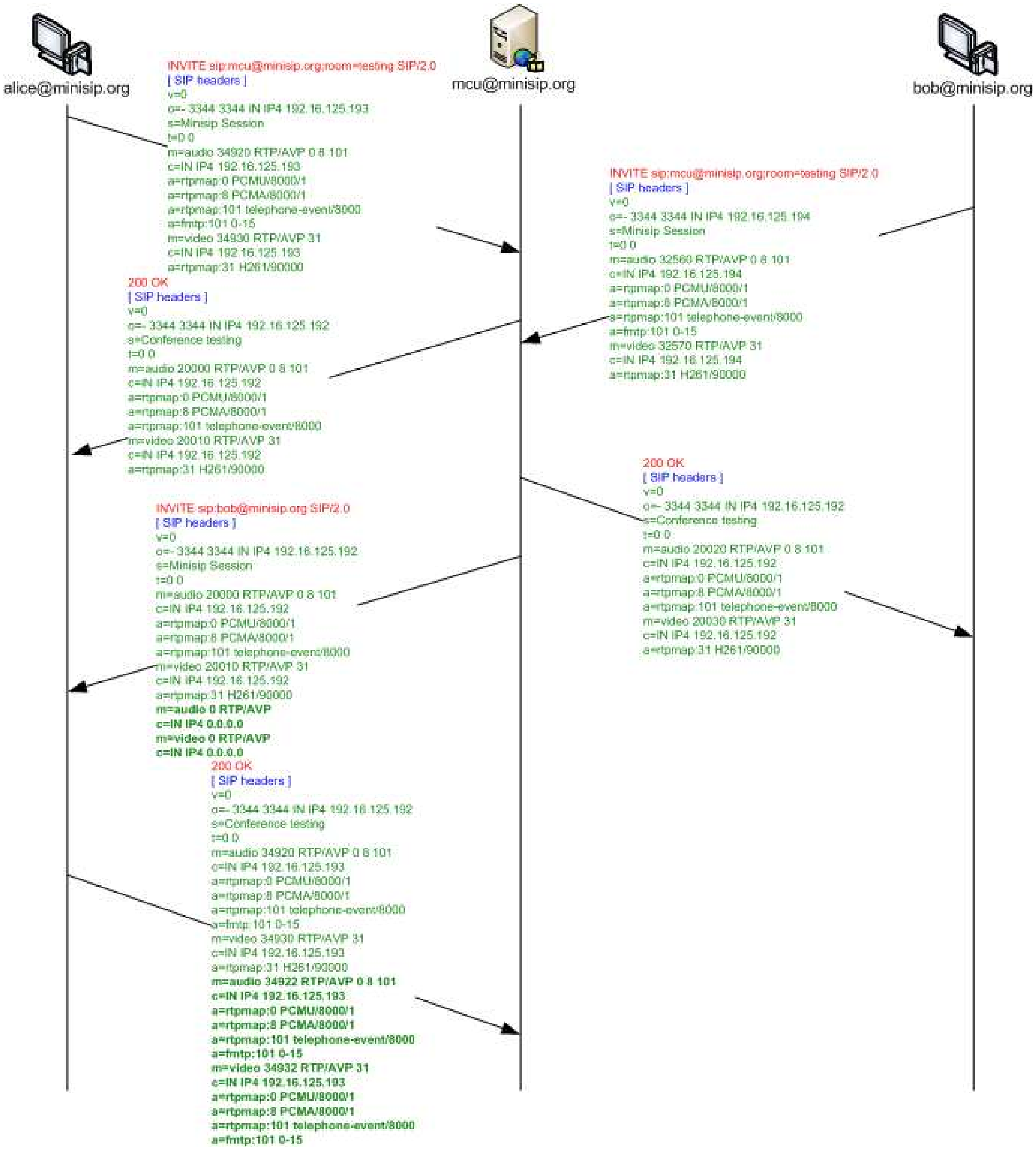
The problem of this solution is that we needed to update the former users in the conference about the changes.

We finally chose to require the new calls to accept, at least, all the media in the room in order to simplify this process.

### 5.3.5. Announcing new participants

In our implementation, when a client joins a conference, it is not announced to the former participants. The MCU starts forwarding the RTP flows and the clients just show or play them. This requires the clients to be capable to receive multiple RTP streams in the same port, but keeps simple the signalling plane.

However, some previous research had been done in this area. In [21], it is proposed to announce the entrance of a new participant in a conference by a re-invite message to all the former participants. They suggest announcing the flows from the new participant as a hold media line offer (using 0.0.0.0 as IP and 0 as port). Then, the client should answer with an SDP announcing where and how (IP, port and RTP media type) it wants to receive these new flows (see Figure 34).



```
INVITE sip:mcu@minisip.org;room=testing SIP/2.0
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.193
s=Minisip Session
t=0 0
m=audio 34920 RTP/AVP 0 8 101
c=IN IP4 192.16.125.193
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 34930 RTP/AVP 31
c=IN IP4 192.16.125.193
a=rtpmap:31 H261/90000
```

```
INVITE sip:mcu@minisip.org;room=testing SIP/2.0
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.194
s=Minisip Session
t=0 0
m=audio 32560 RTP/AVP 0 8 101
c=IN IP4 192.16.125.194
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 32570 RTP/AVP 31
c=IN IP4 192.16.125.194
a=rtpmap:31 H261/90000
```

```
200 OK
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Conference testing
t=0 0
m=audio 20000 RTP/AVP 0 8 101
c=IN IP4 192.16.125.192
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
m=video 20010 RTP/AVP 31
c=IN IP4 192.16.125.192
a=rtpmap:31 H261/90000
```

```
200 OK
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Conference testing
t=0 0
m=audio 20020 RTP/AVP 0 8 101
c=IN IP4 192.16.125.192
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
m=video 20030 RTP/AVP 31
c=IN IP4 192.16.125.192
a=rtpmap:31 H261/90000
```

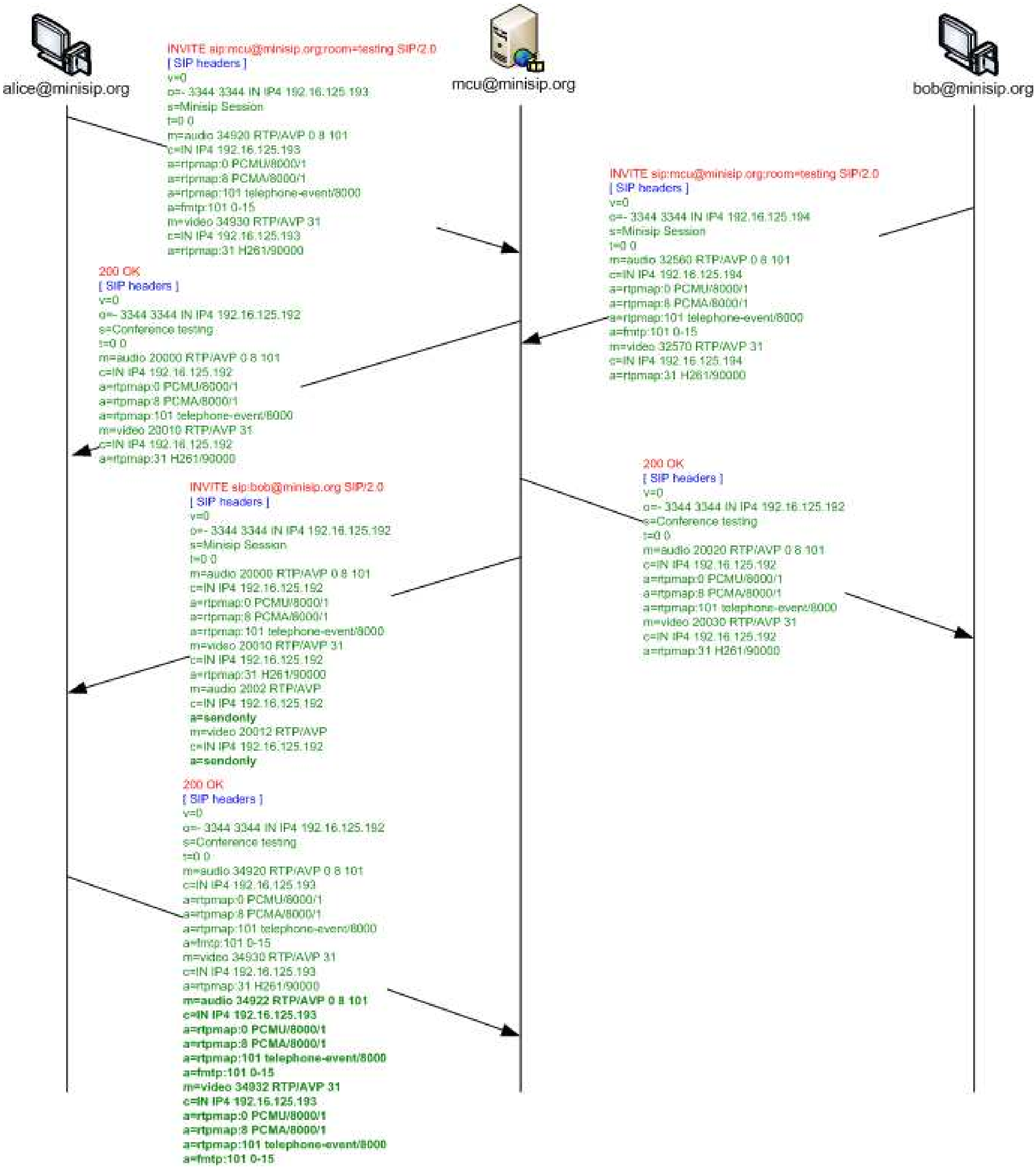
```
INVITE sip:bob@minisip.org SIP/2.0
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Minisip Session
t=0 0
m=audio 20000 RTP/AVP 0 8 101
c=IN IP4 192.16.125.192
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
m=video 20010 RTP/AVP 31
c=IN IP4 192.16.125.192
a=rtpmap:31 H261/90000
m=audio 0 RTP/AVP
c=IN IP4 0.0.0.0
m=video 0 RTP/AVP
c=IN IP4 0.0.0.0
```

```
200 OK
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Conference testing
t=0 0
m=audio 34920 RTP/AVP 0 8 101
c=IN IP4 192.16.125.193
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 34930 RTP/AVP 31
c=IN IP4 192.16.125.193
a=rtpmap:31 H261/90000
m=audio 34922 RTP/AVP 0 8 101
c=IN IP4 192.16.125.193
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 34932 RTP/AVP 31
c=IN IP4 192.16.125.193
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

**Figure 34** MCU re-Inviting operation

Another possibility related to this option would be to use the SDP media attributes *sendonly* and/or *recvonly* (a=sendonly, a=recvonly), as proposed in [22].

On one hand, *sendonly* defines a media flow that the announcer wants to send, but not to receive it. On the other hand, *recvonly* defines a flow the announcer just want to receive, not to send. Back to the example in Figure 34, if these attributes were used, the negotiation would follow as shown in Figure 35.



```
INVITE sip:mcu@minisip.org;room=testing SIP/2.0
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.193
s=Minisip Session
t=0 0
m=audio 34920 RTP/AVP 0 8 101
c=IN IP4 192.16.125.193
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 34930 RTP/AVP 31
c=IN IP4 192.16.125.193
a=rtpmap:31 H261/90000
```

```
INVITE sip:mcu@minisip.org;room=testing SIP/2.0
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.194
s=Minisip Session
t=0 0
m=audio 32560 RTP/AVP 0 8 101
c=IN IP4 192.16.125.194
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 32570 RTP/AVP 31
c=IN IP4 192.16.125.194
a=rtpmap:31 H261/90000
```

```
200 OK
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Conference testing
t=0 0
m=audio 20000 RTP/AVP 0 8 101
c=IN IP4 192.16.125.192
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
m=video 20010 RTP/AVP 31
c=IN IP4 192.16.125.192
a=rtpmap:31 H261/90000
```

```
200 OK
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Conference testing
t=0 0
m=audio 20020 RTP/AVP 0 8 101
c=IN IP4 192.16.125.192
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
m=video 20030 RTP/AVP 31
c=IN IP4 192.16.125.192
a=rtpmap:31 H261/90000
```

```
INVITE sip:bob@minisip.org SIP/2.0
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Minisip Session
t=0 0
m=audio 20000 RTP/AVP 0 8 101
c=IN IP4 192.16.125.192
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
m=video 20010 RTP/AVP 31
c=IN IP4 192.16.125.192
a=rtpmap:31 H261/90000
m=audio 2002 RTP/AVP
c=IN IP4 192.16.125.192
a=sendonly
m=video 20012 RTP/AVP
c=IN IP4 192.16.125.192
a=sendonly
```

```
200 OK
[ SIP headers ]
v=0
o=- 3344 3344 IN IP4 192.16.125.192
s=Conference testing
t=0 0
m=audio 34920 RTP/AVP 0 8 101
c=IN IP4 192.16.125.193
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 34930 RTP/AVP 31
c=IN IP4 192.16.125.193
a=rtpmap:31 H261/90000
m=audio 34922 RTP/AVP 0 8 101
c=IN IP4 192.16.125.193
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
m=video 34932 RTP/AVP 31
c=IN IP4 192.16.125.193
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

**Figure 35** MCU re-Inviting operation with sendonly attributes

Despite these options, receiving multiple streams in the same port make everything easy when dealing with NAT traversal features. Also, it requires for the clients to understand these attributes, which is something not always happens.

### 5.3.6. SRTP key generation for new flows

We modified miniSIP in order to be capable to create new SRTP keys for a RTP flow that was not announced in the MIKEY negotiation. However, we thought about another possibility to solve the same problem.

Instead of changing the client, we could make the MCU to re-invite the client when starting sending a new SRTP flow (see 5.3.5, Announcing new participants). Doing this, the MCU would send a new MIKEY message announcing to the user the new SSRC, so it could create the keys to decrypt the new secured flow.

Nevertheless, we thought the solution we chose to be simplest, although this could be a problem with other MIKEY capable SIP phones.

# 6. Tests and Measurements

Once we finished the software implementation, we wanted to test its efficiency and figure up how many media flows and how much bandwidth it can deal with. We also wanted to realise the processing time difference when using SRTP.

To do so, we needed a source with an output near to 25Mbps, but we did not have any equipment doing that. We solved this problem modifying miniSIP in order to generate byte streams at a certain bit rate to be sent as RTP payloads.

We modified miniSIP to send 2000 RTP packets/second of 1420 bytes of RTP payload each one. The result of this modification was a 22,72Mbps RTP flow, which was very convenient for these tests since it is very similar to HDV video (~25Mbps).

Finally we changed the configuration of the audio devices in miniSIP, to make it read/write to a file instead of the soundcard. The input file was a sequence of 0s and the output one was just an empty and writable file.

## 6.1. Testbed

After that, we set up the testing environment as shown in Figure 36.

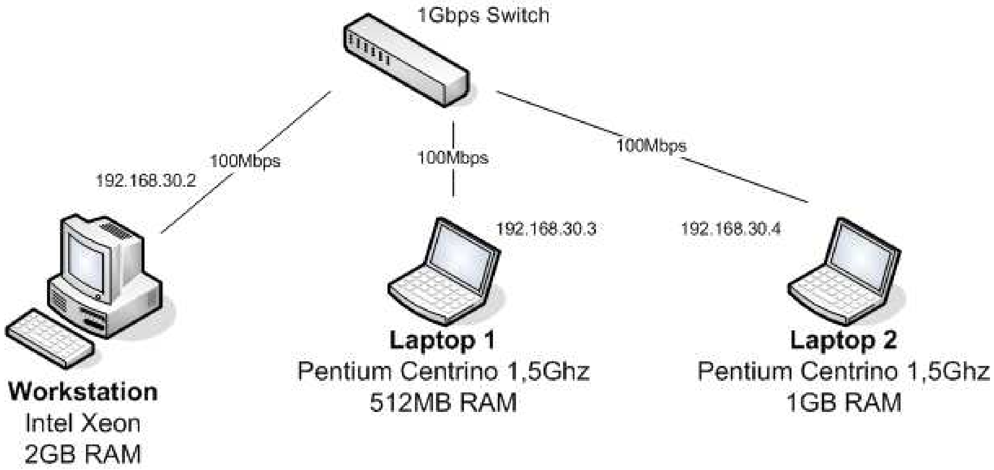


**Figure 36** Testbed topology

In this topology, the MCU runs in Laptop 1 and miniSIP client in all the other computers. From all the clients, a call is made to the same conference room in the MCU. As explained before, the result of this will be the one shown in Figure 37.

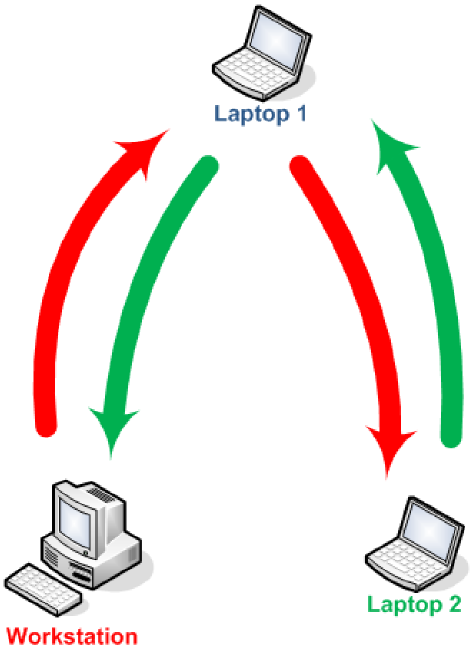


**Figure 37** Flow result

## 6.2. *Measuring tools*

Finally, and before starting the time tests, we needed to add some timestamps in the MCU software. To do so, we used a tool provided by *libmutil*, one of the libraries in miniSIP, called Timestamp.

In every miniSIP based application, a global Timestamp object called *ts* exists. The tool used provides two methods that can be used to measure delays between instructions in the code:

- `save(string name):` adds a timestamp with the text specified.
- `print(string filename):` save the timestamps to the specified file.

To use this tool, we just need to add the following line of code where we want to set a timestamp:

*ts.save(“name”);*

This instruction will generate a new timestamp with the following information:

*name: absolute_time time_since_absolute_timestamp time_since_previous_timestamp*

The absolute time is the UNIX time in seconds. The time since the absolute timestamp and the time since the previous timestamp are expressed in microseconds (μs).

## 6.3. *Measurements*

In our application, we added several save lines to measure the time it takes to execute the following instructions:

- Time to read a RTP packet from an UDP socket (from “Receiving packet…” to “Packet received!”).
- If the packet is encrypted, time to decrypt, authenticate and check the reply protection (from “Unprotecting packet…” to “Packet unprotected!”).
- If the packet has to be sent as a SRTP packet, time needed to encrypt and authenticate it (from “Protecting packet…” to “Packet protected!”).
- Time needed to sent the full (S)RTP packet via an UDP socket (from “Sending packet” to “Packet sent!”).

## 6.4. *Measurement results*

After running the tests using the new software, we got the results detailed in the following lines.

### 6.4.1. Unsecured flows

We started measuring delays in an unsecured environment. We configured miniSIP not to require the call to be secure, so the generated flows were not encrypted. A sample of the results of the time measurements could be the following one:

```
Receiving packet...:   1213102654:   980448    101
Packet received!:      1213102654:   980451    3
Sending packet...:     1213102654:   980461    10
Packet sent!:          1213102654:   980503    42
Receiving packet...:   1213102654:   980579    76
Packet received!:      1213102654:   980582    3
Sending packet...:     1213102654:   980593    11
Packet sent!:          1213102654:   980630    37
Receiving packet...:   1213102654:   980704    74
Packet received!:      1213102654:   980707    3
Sending packet...:     1213102654:   980717    10
Packet sent!:          1213102654:   980801    84
Receiving packet...:   1213102654:   980850    49
Packet received!:      1213102654:   980854    4
Sending packet...:     1213102654:   980864    10
Packet sent!:          1213102654:   980883    19
Receiving packet...:   1213102654:   980983    100
Packet received!:      1213102654:   980986    3
Sending packet...:     1213102654:   980999    13
Packet sent!:          1213102654:   981062    63
```

As can be seen in the sample, for each packet the MCU receives, it is forwarded to the other participant in the conference. This is exactly the behaviour expected in the environment we set up.

Looking at the results, we can make some calculations in order to figure up how much bandwidth the MCU can deal with.

- Average delay introduced in each packet

$$t = \frac{\dfrac{(3+10+42)+(3+11+37)+(3+10+84)+(4+10+19)+(3+13+63)}{2}}{5} = 31{,}5\mu s / packet$$

- Maximum theoretical throughput

$$BW = \frac{1packet}{31{,}5\mu s} \cdot \frac{1420bytes}{1packet} \cdot \frac{8bits}{1byte} = 360{,}63Mbps$$

Once we calculate the theoretical throughput, if we define a certain bandwidth to be sent from each user, it is easy to find how many users can be supported by the MCU. If we assume each user is just sending 22,72Mbps, we can find the number of users supported by our MCU in this scenario:

$$\#\,flows = (\#users)^2$$

$$\frac{360{,}63Mbps}{22{,}72\dfrac{Mbps}{user}} = N^2; N = \sqrt{\frac{360{,}63}{22{,}72}} \approx 4users$$

Unfortunately, and due to the lack of computers with a 1Gbps Ethernet interface, we could not verify neither if these calculations were right nor if these values were the same with more users in the system.

### 6.4.2. SRTP secured flows

After taking measures in the unsecured environment, we switched on security on all miniSIP clients in order to secure sent media flows. We got the flow schema shown in Figure 38.

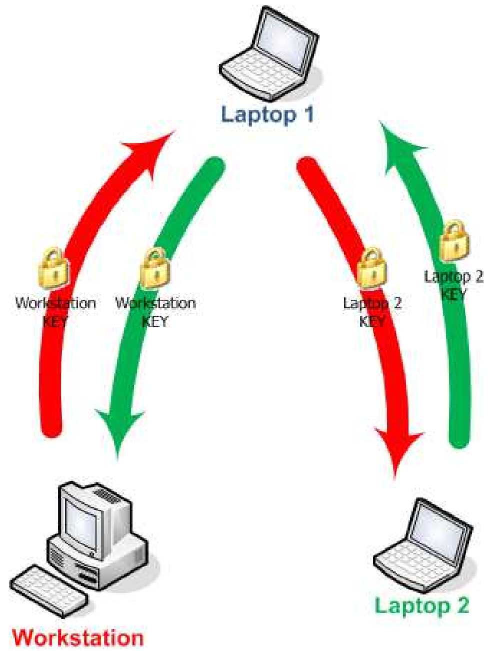


**Figure 38** Flow result in a secure scenario

A sample of the time measurement results obtained in this scenario would be the following one:

```
Receiving packet...:    1213102908:   447898        6
Packet received!:       1213102908:   447906        8
Unprotecting packet...: 1213102908:   447914        8
Packet unprotected!:    1213102908:   447963        49
Protecting packet...:   1213102908:   447968        5
Packet protected!:      1213102908:   448041        73
Sending packet...:      1213102908:   448043        2
Packet sent!:           1213102908:   448055        12
Receiving packet...:    1213102908:   448101        46
Packet received!:       1213102908:   448105        4
Unprotecting packet...: 1213102908:   448113        8
Packet unprotected!:    1213102908:   448163        50
Protecting packet...:   1213102908:   448167        4
Packet protected!:      1213102908:   448236        69
Sending packet...:      1213102908:   448237        1
Packet sent!:           1213102908:   448250        13
Receiving packet...:    1213102908:   448256        6
Packet received!:       1213102908:   448260        4
Unprotecting packet...: 1213102908:   448268        8
Packet unprotected!:    1213102908:   448318        50
Protecting packet...:   1213102908:   448322        4
Packet protected!:      1213102908:   448371        49
Sending packet...:      1213102908:   448395        24
Packet sent!:           1213102908:   448445        50
```

If we just look at the times of the protection and unprotection process of the packets, it is possible to calculate the average time to unprotect or protect a SRTP packet:

$$t_{unprotect\,/\,protect}=\frac{\dfrac{\frac{49+73}{2}+\frac{50+69}{2}+\frac{50+49}{2}}{2}}{3}=32{,}5\mu s$$

If we use now the forwarding time obtained in the unsecured scenario, we can calculate the theoretical throughput of the system running with SRTP flows and the maximum number of supported users, defining that each user just sends 22.72Mbps:

$$BW=\frac{1packet}{31{,}5\mu s+32{,}5\mu s}\cdot\frac{1420bytes}{1packet}\cdot\frac{8bits}{1byte}=177{,}55Mbps$$

$$\frac{177{,}55Mbps}{22{,}72\dfrac{Mbps}{user}}=N^2;N=\sqrt{\frac{177{,}55}{22{,}72}}\approx 3users$$

### 6.4.3. Alternative measurements with unsecured flows

We also had the chance to make some tests in a more suitable scenario (shown in Figure 39). Due to some problems related to the implementation in the lasts versions of the Packet Reflector, it did not work properly in a multi-core computer.

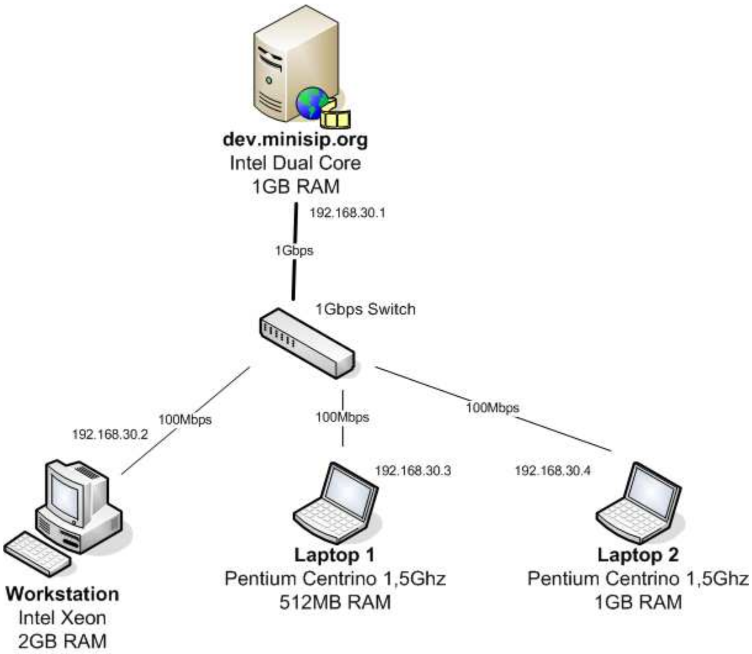


**Figure 39** Alternative testbed

Anyway, we used an old and inefficient version of the Packet Reflect that worked in multi-core machines to take more measurements. A sample of these would be the following one:

```
Receiving packet...    : 1212654951:  818979      15
Packet received!       : 1212654951:  818984      5
Sending packet...      : 1212654951:  818997      13
Packet sent!           : 1212654951:  819012      15
Sending packet...      : 1212654951:  819018      6
Packet sent!           : 1212654951:  819026      8
Receiving packet...    : 1212654951:  819041      15
Packet received!       : 1212654951:  819045      4
Sending packet...      : 1212654951:  819059      14
Packet sent!           : 1212654951:  819067      8
Sending packet...      : 1212654951:  819073      6
Packet sent!           : 1212654951:  819087      14
Receiving packet...    : 1212654951:  819102      15
Packet received!       : 1212654951:  819106      4
Sending packet...      : 1212654951:  819120      14
Packet sent!           : 1212654951:  819129      9
Sending packet...      : 1212654951:  819134      5
Packet sent!           : 1212654951:  819143      9
```

As can be seen in the sample, for each packet the MCU receives, it is forwarded to the other 2 participants in the conference. This is exactly the behaviour expected in the environment we set up (shown in Figure 40).

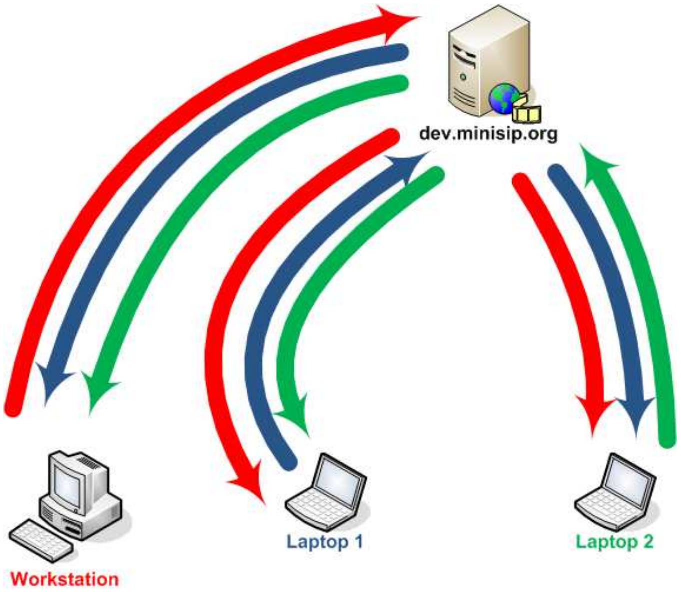


**Figure 40** Flow result

Looking at the results, we can make some calculations in order to figure up how much bandwidth the MCU can deal with in this situation.

- Average delay introduced in each packet

$$t = \frac{5 + \left(\frac{13+15+6+8}{2}\right) + 4 + \left(\frac{14+8+6+14}{2}\right) + 4 + \left(\frac{14+8+6+14}{2}\right)}{3} = 24{,}5\mu s / packet$$

- Maximum theoretical throughput

$$BW = \frac{1 packet}{24{,}5\mu s} \cdot \frac{1420 bytes}{1 packet} \cdot \frac{8 bits}{1 byte} = 463{,}67 Mbps$$

Once we calculate the theoretical throughput, and defining a certain bandwidth to be sent from each user, it is easy to find how many users can be supported by the MCU. If we assume each user is just sending 22,72Mbps again, we can find the number of users supported by our MCU in this scenario:

$$\frac{463{,}67 Mbps}{22{,}72 \frac{Mbps}{user}} = N^2; N = \sqrt{\frac{463{,}67}{22{,}72}} \approx 5 users$$

Unfortunately, and due to the lack of computers with a 1Gbps Ethernet interface, we could not check if these calculations were right or if we could add more users to the conference.

We could neither test this scenario with secured flows, since the design of the old version of the Packet Reflector was not ready to add the parameters needed for it. However, we could make some theoretical calculations based on the measurements took in 6.4.2 (SRTP secured flows).

We applied the average time to unsecure or secure a packet and other times obtained in 6.4.2 (SRTP secured flows) to this scenario, so we obtained the average time to deal with a packet:

$$t_{packet} = 32{,}5 + 24{,}5 = 57\mu s$$

With this time is easy now to calculate the theoretical maximum throughput and the number of users the system can handle:

$$BW = \frac{1 packet}{57\mu s} \cdot \frac{1420 bytes}{1 packet} \cdot \frac{8 bits}{1 byte} = 199{,}3 Mbps$$

$$\frac{199{,}3 Mbps}{22{,}72 \frac{Mbps}{user}} = N^2 ; N = \sqrt{\frac{199{,}3}{22{,}72}} \approx 3 users$$

Taking into account these calculations were just theoretical and that we mixed times obtained using either a good computer but bad software or better software and a slow computer, we feel that the use of security has an affordable cost in terms of computing time, unless a weak computer is used as the MCU.

# 7. Final conclusions

This chapter of conclusions will focus on the signalling layer and security issues of my implementation, since they were the main targets of this study.

During the early stages of the design and the implementation of the MCU we took some decisions that now do not seem to have been the best ones. I will try to explain our vision in the next paragraphs.

In our implementation, we assumed that the SIP client was able to receive several RTP streams in the same UDP port. Our decision was done thinking about miniSIP, which can do that, but later tests showed that most other SIP clients (such as Ekiga or hardware SIP phones) can not do that. This is a problem for our software, since these clients will be able to play only one of the flows received and the user will be able to communicate just with one of the participants in the conference.

The announcement of new participants joining the conference is linked to the multiple streams issue. In our implementation, the MCU does not inform the former participants that a new user joined the conference, it just starts forwarding the flows coming from this new user to the other ones. As explained in 5.3.5 (Announcing new participants), using re-invite messages and announcing new media flows in the SDP with the *sendonly* attribute seems to be the optimal solution to solve this problem, since the SIP clients would not need to support multiple streams in the same port. Also, we avoid the problem of ensuring everyone in a conference can receive all types of media sent (5.1.3, SDP media dealing), since the clients would reject all re-invites with SDP offers including unsupported media types.

Moreover, thinking about MIKEY integration in this last solution, MIKEY messages could be sent in every re-invite, so the new RTP SSRCs would be announced in the SDP. By this way, SRTP keys could be generated before any media is sent and we would not need to modify miniSIP to read the SSRC from the first received packet.

Despite all these issues appeared during the development of this thesis, we built a working system. All the knowledge acquired will be used for further implementations within the HDVIPER project, taking into account these conclusions.

If the test results are analyzed, one could think that the MCU could be a limitation in terms of bandwidth. That is probably true, but the lack of multicast in most of the public networks drove us to choose this system. The good point about this is that it can also be used in a multicast environment to allow non-multicast capable users to join a conference (see Figure 41). Furthermore, in this situation the MCU could also act as the central signalling point needed in multicast conferences.

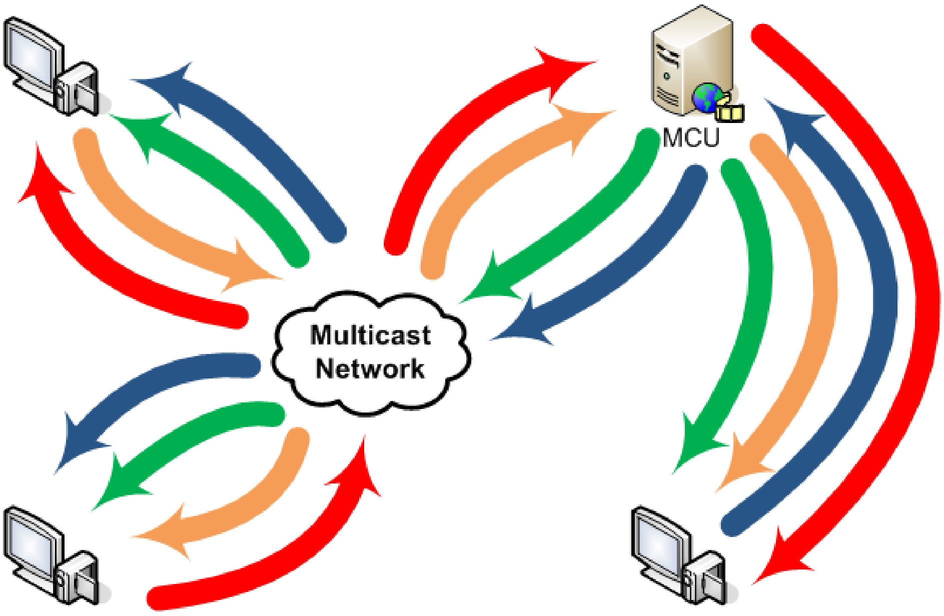


**Figure 41** MCU in a multicast network

Also, we showed that using a forwarding MCU is convenient for videoconferencing, since the delay introduced by this element is very short. Although we didn't implement them, we found some solutions to solve the problems appeared with the signalling of different types of media.

We also discovered that the weakness of this system is not in the software itself but in the bandwidth available in the computer running the MCU. We made some calculations to prove that the maximum number of users the system can handle by a 1Gbps full duplex Ethernet is 7 users:

$$BW_{upload} = (\#users)^2 \cdot BW_{user} - \#users \cdot BW_{user}$$

$$BW_{7users} = 7^2 \cdot 22{,}72Mbps - 7 \cdot 22{,}72Mbps = 954{,}14Mbps$$

$$BW_{8users} = 8^2 \cdot 22{,}72Mbps - 8 \cdot 22{,}72Mbps = 1272{,}32Mbps$$

Finally, although we only could test the new software with two users and a slow computer due to the problems in the Packet Reflector, we proved that using SRTP to secure media flows can be used to protect high bandwidth flows when using an MCU, since the delay added by this feature was short enough for videoconferencing. Moreover, if we think using a powerful computer to run the MCU or even using AES hardware en/decrypters, the delay would be shorter, more users could be supported and the performance could be better than in our tests.

# 8. References

[1] Wikipedia. *Videoconferencing*. Wikipedia <http://en.wikipedia.org/wiki/Videoconferencing>. Published 12-2007, read 12-2-2008.

[2] López Rubio, Javier; Oller Arcas, Antoni. *MCU para multiconferencias en alta definición*. Universitat Politècnica de Catalunya, Escola Politècnica Superior de Castelldefels, 2007.

[3] Alcober Segura, Jesús; López Rubio, Javier; Oller Arcas, Antoni; *High-Definition Video Conferencing MCU with SIP signaling capabilities*. Department of Telematics Engineering, Universitat Politècnica de Catalunya / Fundació I2cat, 2007.

[4] Anagrius, Jimmy; Bishaj, Blerta; Ngelja, Leonard James; Rajwani, Sanjay. *ComCloser* < http://www.tslab.ssvl.kth.se/csd/projects/0727/> Read 12-2-2008.

[5] Rincon Rivera, David. *Intensificació en Serveis Telemàtics: Protocols IP relacionats amb multimèdia*. Universitat Politècnica de Catalunya, Escola Politècnica Superior de Castelldefels, February 2007.

[6] Wikipedia. *H.323*. Wikipedia <http://en.wikipedia.org/wiki/H.323>. Published 10-2006, read 15-2-2008.

[7] Rosenberg, J. and others. *SIP: Session Initiation Protocol*. Internet Engineering Task Force, RFC 3261, 2002.

[8] Handley, M; Jacobson, V; Perkins, C. *SDP: Session Description Protocol*. Internet Engineering Task Force, RFC 4566, 2006.

[9] Wikipedia. *Session Description Protocol*. Wikipedia <http://en.wikipedia.org/wiki/Session_Description_Protocol>. Published 7-2004, read 19-2-2008.

[10] Schulzrinne, H. and others .*RTP: A Transport Protocol for Real-Time Applications*. Internet Engineering Task Force, RFC 3550, 2003.

[11] Schulzrinne, H; Casner, S. *RTP Profile for Audio and Video Conferences with Minimal Control*. Internet Engineering Task Force, RFC 3551, 2003.

[12] *HDV format. Main Specifications* <http://www.hdv-info.org/HDVSpecifications.pdf>. Read 22-2-2008.

[13] Kaufman, Charlie; Perlman, Radia; Speciner, Mike. *Network Security. Private Communication in a PUBLIC World*. Prentice Hall PTR, Second Edition, 2002.

[14] Bilien, Johan. *Key Agreement for Secure Voice over IP*. Royal Institute of Technology (KTH), Center for Wireless Systems and Telecommunication Systems Laboratory, 2003.

[15] Wikipedia. *S/MIME*. Wikipedia < *http://en.wikipedia.org/wiki/S/MIME*>. Published 11-2004, read 24-2-2008.

[16] Bilien, Johan; Eliasson, Erik; Orrblad, Joachim; Vatn, Jon-Olov. *Secure VoIP: call establishment and media protection*. Royal Institute of Technology (KTH), 2005.

[17] Svensson, Mikael. *Countering VoIP Spam: Up-Cross-Down Certificate Validation*. Royal Institute of Technology (KTH), Telecommunications System Laboratory, 2007.

[18] Orrblad, Joachim. *Alternatives to MIKEY/SRTP to secure VoIP*. Royal Institute of Technology (KTH), Telecommunications System Laboratory, 2005.

[19] Baugher, M. and others. *The Secure Real-time Transport Protocol (SRTP)*. Internet Engineering Task Force, RFC 3711, 2004.

[20] Arkko, J. and others. *MIKEY: Multimedia Internet KEYing*. Internet Engineering Task Force, RFC 3830, 2004.

[21] Rosenberg, J. and others. *Models for Multi Party Conferencing in SIP*. Internet Engineering Task Force, Internet Draft, 2001.

[22] Rosenberg, J. and Schulzrinne, H. *An Offer/Answer Model with the Session Description Protocol (SDP)*. Internet Engineering Task Force, RFC 3264, 2002.

[23] *miniSIP Web Page* < http://www.minisip.org >. Last visit on 28-5-2008.

[24] *Subversion Web Page* < http://subversion.tigris.org/ >. Last visit on 28-5-2008.

[25] *SQLite Project Web Page* < http://www.sqlite.org/ >. Last visit on 11-6-2008.

[26] *SIP Express Router Web Page* < http://www.iptel.org/ser/ >. Last visit on 13-6-2008.

# 9. Appendix A – miniSIP

miniSIP is an Open-Source SIP User Agent (also known as SIP phone or Internet Phone) oriented to provide security to the calls done over Internet.

It is designed in different code modules, called libraries, that join together to build a complete SIP phone. These libraries can be used separately to create new applications and that is what we did to develop the MCU.

More information can be found on the miniSIP project webpage [23].

## *9.1. miniSIP libraries used by the MCU*

In order to make the new MCU application works, we used the following miniSIP libraries. All of them need to be compiled to use the MCU.

- ***libmutil***: cross platform support for threads, semaphores, mutex and other utilities. Used by all the other libraries.
- ***libmnetutil***: cross platform support for threads and networking utilities. Used by *libmsip* and *libminisip*.
- ***libmcrypto***: cross platform support for security utilities, such as AES encryption/decryption or SmartCard based authentication. Used by *libminisip*.
- ***libmstun***: support for NAT traversal using a STUN server. Used by *libminisip*.
- ***libmikey***: support for MIKEY protocol. Used by *libminisip*.
- ***libmsip***: full state SIP stack and utilities to create simple SIP applications. Used by *libminisip*.
- ***libminisip***: full SIP phone without a GUI.

In the development of the MCU, we used directly tools from *libmutil*, *libmikey*, *libmsip* and *libminisip*. However, all the other libraries are needed for the right behaviour of the mentioned ones.

## *9.2. Getting miniSIP*

miniSIP uses Subversion (also known as SVN) [24] as a version control and repository system, so to get the latest version of the code a SVN client is needed.

In Linux, we can use the terminal SVN client to get it:

```
svn co svn://svn.minisip.org/minisip/trunk
```

Once it finishes downloading it, the source code of the latest version of miniSIP will be in the local computer.

## 9.3. Compiling miniSIP

Once we obtained the source code via SVN, we need to compile the libraries to use them in the MCU. To do so, the following compiling tools are needed:

- *g++*
- *make*, *automake* and *autoconf*
- *libtool*

A part of that, miniSIP has some dependency on other packages that must be solved before installing it:

- *libglademm* and *libgtkmm*, used in the GTK GUI.
- *libssl-dev* and *openssl*, used for security.
- *libltdl-dev*, used for the plugin system.

Before starting compiling, some settings must be written in the .bashrc file of the user. To do so, open the /home/user/.bashrc file and add the following lines at the end:

```
export ACLOCAL_FLAGS="-I /home/user/share/aclocal"
export LD_LIBRARY_PATH=/home/user/lib
export CPPFLAGS=-I/home/user/include
export LDFLAGS=-L/home/user/lib
export PKG_CONFIG_PATH=/home/user/lib/pkgconfig/
```

After that, run the next command to update the changes:

```
bash
```

And create the following folders (use the *mkdir* command):

- /home/user/share/aclocal
- /home/user/lib/pkgconfig
- /home/user/include
- /home/user/bin

Finally, the computer is ready to start compiling miniSIP libraries. Run the following commands in a terminal and wait for it to end:

```
cd libmutil
for i in libmutil libmnetutil libmcrypto libmstun libmikey libmsip libminisip
minisip ; do cd ../$i; ./bootstrap && ./configure --enable-debug --disable-gconf
--enable-gtk --enable-textui --enable-video prefix=/home/$USER
&& make && make install ; done
```

By this command we configure and compile all the libraries needed by the MCU and the two GUIs (text based and GTK based) for the SIP phone. We enable the debug mode, the two GUIs and the video support, and we disable the configuration through the GConf configuring tool.

# 10. Appedix B – Using the new MCU application

In order to use the new MCU application, it is first needed to compile the miniSIP libraries listed in 9.1 (miniSIP libraries used by the MCU). To do so, follow the instructions in 9.3 (Compiling miniSIP).

Once the miniSIP libraries are correctly compiled and installed, the following packages should be installed, since they are dependencies of the Packet Reflector:

- *libboost-devel*
- *libboost-thread*

As soon as these packages are correctly installed, the system is ready to compile the MCU application. To do so, the Makefile provided with the software can be used, so it is just needed to run *make* in a terminal.

Once it is compiled, it can be run by the following command:

```
./bin/mcu –u [SIP URI] –p [SIP PORT] -i [network interface]
```

This will run an instance of the software, using the SIP URI specified, listening on the SIP port specified and using the network interface specified.

# 11. Appendix C – Terms and abbreviations

| | |
|---|---|
| *3GPP* | 3rd Generation Partnership Project |
| *ADSL* | Asymmetric Digital Subscriber Line |
| *AES* | Advanced Encryption Standard |
| *CA* | Certificate Authority |
| *DH* | Diffie-Hellman |
| *DNS* | Domain Name Server |
| *GSM* | Groupe Spécial Mobile |
| *HMAC* | keyed-Hash Message Authentication Code |
| *HTTP* | Hyper-Text Transfer Protocol |
| *IETF* | Internet Engineering Task Force |
| *IKE* | Internet Key Exchange |
| *IP* | Internet Protocol |
| *IPSec* | Internet Protocol Security |
| *ISDN* | Integrated Services Digital Network |
| *ISAKMP* | Internet Security Association and Key Management Protocol |
| *ITU* | International Telecommunication Union |
| *MAC* | Message Authentication Code |
| *MCU* | Multipoint Conference Unit |
| *MIKEY* | Multimedia Internet Keying |
| *MPEG* | Moving Pictures Experts Group |
| *MTU* | Maximum Transfer Unit |
| *NASA* | National Aeronautics and Space Administration |
| *NAT* | Network Address Translation |
| *NTP* | Network Time Protocol |
| *NTSC* | National Television System Committee |
| *PAL* | Phase Alternating Line |
| *PCM* | Pulse Code Modulation |
| *PDA* | Personal Digital Assistant |
| *PKE* | Public Key Encryption |
| *PKI* | Public Key Infrastructure |
| *PSK* | Pre-Shared Key |
| *PSTN* | Public Switched Telephone Network |
| *RFC* | Request for Comments |
| *RTCP* | Real-time Transfer Control Protocol |
| *RTP* | Real-time Transfer Protocol |
| *SCTP* | Stream Control Transmission Protocol |
| *SDP* | Session Description Protocol |
| *SHA-1* | Security Hash Algorithm 1 |
| *SMTP* | Simple Mail Transfer Protocol |
| *SIP* | Session Initiation Protocol |
| *SIP-URI* | Session Initiation Protocol Universal Resource Identifier |
| *SRTCP* | Secure Real-time Transfer Control Protocol |
| *SRTP* | Secure Real-time Transfer Protocol |
| *SVN* | Subversion |
| *TCP* | Transmission Control Protocol |
| *TEK* | Transport Encryption Key |
| *TGK* | Transport Generation Key |
| *TLS* | Transport Layer Security |
| *UA* | User Agent |
| *UAC* | User Agent Client |
| *UAS* | User Agent Server |
| *UDP* | User Datagram Protocol |
| *UHF* | Ultra High Frequency |
| *VoIP* | Voice over Internet Protocol |