Voice over IP Measurements

Radu Dudici Ruscior

LYNGBY 2002
IMM-THESIS-2002-75

Printed by IMM, DTU

Preface

This M.Sc. thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Systems Engineering. The work has been carried out in the period from 1st of March 2002 to 29th of November 2002 at the department of Informatics and Mathematical Modeling, Technical University of Denmark. The work has been supervised by Professor Steen Pedersen and Professor Robin Sharp. The thesis has been carried out in collaboration with Gavnholt Communications under supervision of Uffe Gavnholt.

Also a tool, called VIPSim has been obtained as a result of this project. Although this tool has been used to obtain the results commented in this thesis its source code is not listed here for commercial reasons.

I would like to thank my supervisors Steen Pedersen, Robin Sharp and Uffe Gavnholt for their assistance and guidance.

I would also like to thank my future wife, Andreea, for her big support.

Lyngby - November , 2002

Radu Dudici Ruscior

Abstract

The Quality of Service is very important for any user, related to Voice over IP or any other multimedia application. This thesis presents a solution to develop a tool able to measure this parameter in order to detect upcoming bottleneck problems for a future installation of a VoIP system and to troubleshoot existing VoIP systems. The tool's name is VIPSim.

VIPSim is meant to be used to provide extensive information about performance, capacity and problems in an H.323 network. It is also used for alarms, planning and documentation.

The thesis is structured in 7 sections.

Section 1 presents the general problem of QoS in a VoIP system and shows a possible solution, which is VIPSim.

Section 2 presents the architecture of VIPSim. It describes the two components of VIPSim, which are the Sender and the Reflector. It also shows what VIPSim does and how it does (referring to measurements). It addresses to users of VIPSim in systems troubleshootings.

Section 3 goes deeper into VIPSim's implementation, describing its internal design as well as the description of the implementation of some important operations. It addresses to programmers and developers.

Section 4 describes some aspects regarding optimizations that were used in VIPSim's implementation. This section also gives the reader ideas about some algorithms used in VIPSim's implementation.

Section 5 shows tests and results obtained from measurements done with VIPSim. Some of them are meant to determine VIPSim's limitations and requirements.

Section 6 contains general conclusions.

Section 7 contains a glossary with terms used in this thesis, and some VoIP terminology.

Keywords VoIP, Packets, Round Trip Delay, H.323, Jitter, Network, Measurements, Calls

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This section gives a description of Quality of Service in Voice over IP, and proposes a solution for measuring and monitoring this parameter, which is VIPSim.

1.1 Voice over IP and Quality of Service

As the network technologies advance, new other technologies are developed. This is also the case of voice over internet protocol or simply VoIP. It started as a way of cheap long distance calls, becoming in short time a multimedia technology that is constantly developing.

VoIP is the technology in which the analogue signal from a standard telephone is converted into bits, grouped in packets of data, and transmitted over digital lines. In other words, using VoIP, instead of talking on the telephone, you use the computer and through the Internet you talk with the other party.

This has some advantages, like low cost of long distance calls, due to low cost of internet. Another advantage is that wherever there is an Internet connection there is a possibility to transmit voice over the Internet, meaning that there is the possibility to talk with someone. Talking with someone like talking on the phone is not the only feature of this technology. Transmitting voice over digital lines may also be used for listening to music and live radio broadcasts, and even more, this is an important step in going further to transmitting video over the Internet.

As this technology develops a new problem arises, the quality of the service provided by VoIP or simply QoS. Of course in the beginning when this technology just arrived, the quality was pour but now when we have broadband networks with high speed data transfer these services get better and better.

Quality of service (QoS) can be defined as the ability of a system, including software, devices and the network, to provide guaranteed and better service regarding transmission of data. For VoIP this is based on the human perception of voice that is sensitive to delays. From a user's point of view quality means clear, continuous and real-time sound. From an application's point of view this is translated into no delays, no packets lost on the network and enough bandwidth.

For a VoIP system the parameters that define the QoS [7] are:

Bandwidth - is the speed of transmitting data measured in bits per second. A low bandwidth as well as a network connection that has the data traffic close to its limit of bandwidth, will affect the quality of voice. A standard voice call needs a bandwidth of 64kbit/s. Using different coders the requirements for bandwidth may vary.

Delay - is the difference between the receiving time and the sending time of the voice data. There are many sources of delay [6].

- *Coders delay* - time needed by the DSP module to compress a voice signal.
- *Packetization delay* - time needed to fill in the packet payload with the compressed data and to attach the headers.
- *Queuing/Buffering delay* - time needed to queue the packets to be transmitted on the network.
- *De-jittering delay* - the time used for the receiver to accumulate packets into buffers in order to reduce the jitter delay.
- *Network delays* - delays that appear due to transmission on the physical link as well as the interferences of network devices like routers, switches, firewalls, e.t.c.

Jitter - is the variation in the delay of received packets [6] or an estimation of statistical variance of the packets interarrival time [2]. Packets are sent at a constant rate and they are expected to arrive at the same rate, but due to network congestions for example this rate may vary.

Link quality - the network needs to provide besides a fast connection also an error-free connection. This is measured in bit-error-rate. Some bit errors can be corrected at the receiver by using special mechanisms.

Packets out of order - they are the voice packets that do not arrive in the order they were sent. These packets are discarded and the voice information is missing.

Lost packets - they are packets that are lost on the network and do not arrive at the destination.

Due to ITU-T recommendation G.114 the accepted delay is between 0 - 150 ms for all users. Between 150 - 400 ms is acceptable provided that administrators are aware of the transmission time and its impact on the transmission quality of user applications. Above 400 ms the delay is not acceptable.

The delay may cause two problems: echo and talkers overlap. Echo appears when the sent voice signal is reflected back by the equipment on the other far end and played back to the sender. This becomes a serious problem when the round trip delay is greater than 50 ms. Talker overlap becomes significant problem when the one-way delay is greater than 250 ms [10].

If the jitter is excessive it may introduce interruptions in the played sound. This effect may be minimized using some buffers at the receiver. These buffers accumulate a certain amount of packets before releasing them to be played, and the process is called de-jittering. The process of de-jitter also adds some delay.

The discarded packets together with the lost packets introduce noise or make the words to sound like being chopped. This can be corrected at the receiver by using a mechanism that tries to approximate the missing data, but if the number of missing packets is too big, the sound will be affected. An accepted value for lost packets is around 5-10 % from the total number of packets.

1.2 VoIP system - H.323 recommendation

1.2.1 H.323 - Packet-based multimedia communication systems

H.323 is a recommendation from International Telecommunication Union (ITU-T) that contains protocols and procedures used by terminals and other entities that provide multimedia communications services over Packet Based Network (PBN). H.323 entities may provide video, audio and/or data communications [1]. As an example a VoIP application may be implemented based on this recommendation, but is not necessary to provide the video functionality. The audio capability is mandatory for all H.323 entities.

An H.323 system consists of several components listed below. H.323 recommendation describes these components and defines how they communicate. These components are:

- *H.323 terminals* - establish calls between two or more parties.
- *Gatekeepers* - provide admission control and address translation services.
- *Gateways* - translation between different formats and communication procedure
- *Multipoint control unit* (MCU) - provide support for multipoint conferences.
- *Multipoint controller* (MC) - also provide support for multipoint conferences.
- *Multipoint processor* (MP) - also provide support for multipoint conferences.

There are H.323 entities that have the functionality of more than just one component listed above. For example a simple terminal may become a MCU, MC or MP controlling in this way a conference between several parties.

H.323 recommendation is a collection of several other recommendations. As may be seen in the figure 1.1 these recommendations are:

H.225.0 - Call signalling protocols and media stream packetization for packet based multimedia communication systems.
H.245 - Control protocol for multimedia communication.
H.261, H.263 - recommendations regarding video coders.
G.711, G.722, G.723, G.728, G.729 - recommendation regarding audio coders.

The diagram in the figure 1.1 represents a general description of an H.323 terminal.

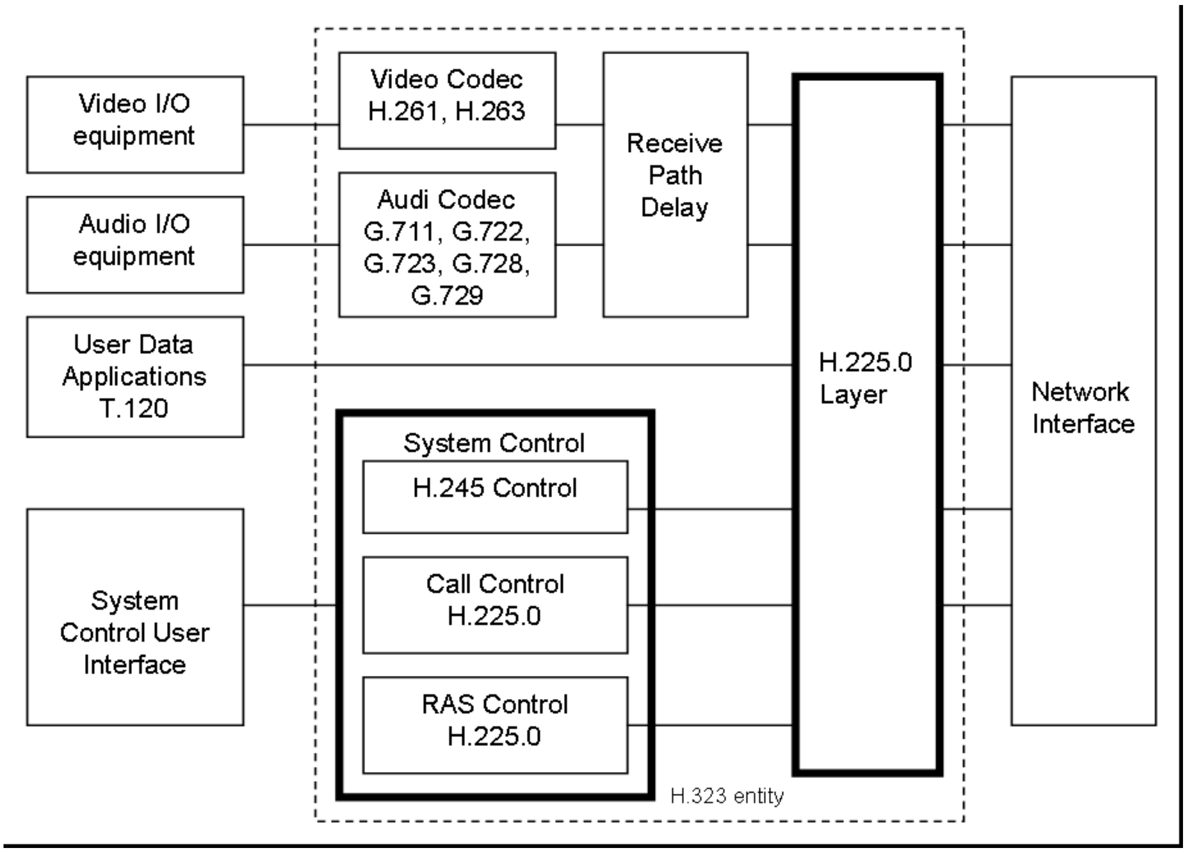


Figure 1.1: A general H.323 terminal.

The components of this terminal are:

Receive Path Delay - includes delay added to the media stream for synchronization.

System Control - provides signalling for call control, capability exchange, commands, indications, open channels.

H.225.0 Layer - formats the transmitted signals (video, audio, data and control) into messages for output to network interface.

Video and Audio codec - encodes/decodes video/audio signal from equipment.

The procedures of the System Control component are defined in the recommendations H.225.0 and H.245, and they provide signalling for a proper operation of H.323 entities. They contain messages for registration and admission to a gatekeeper, messages for call control (call establishment and call ending), messages for capability exchange, opening channels and other commands and indications [1].

Call control messages defined in recommendation H.225.0 are implemented based on the recommendation Q931 [4]. This recommendation defines the format of the call control messages. These messages are used for establishing or ending calls and some of them are listed in appendix A.1. For a complete list please see the recommendation H.225.0 [2].

The recommendation H.225.0 defines also RAS (Registration, Admission and Status) messages that are used in communication with the gatekeepers. These messages do not use the format defined in Q.931. Some of them are also listed in appendix A.1. For a complete list of RAS messages please see the recommendation H.225.0 [2]:

The recommendation H.245 defines messages and procedures for negotiations that take place after the call was establish. These messages are used for receiving and transmitting of capabilities, preferred modes as well as logical channel signalling and other control and indications messages. Some of these messages are listed in appendix A.1. For a complete list, please see the recommendation H.245 [3]. Also, acknowledge messages are defined in order to ensure reliable communication.

All the messages defined in recommendations H.225.0 and H.245 are encoded using the ASN1 syntax.

1.3 Purpose of the application - VIPSim

1.3.1 General description

The purpose of this project is to develop an algorithm and to build an application, based on this algorithm that is able to measure the QoS that can be offered by a system. The name of this application is VIPSim and it stands for Voice over

IP Simulations. The measured QoS shows the upcoming performances of a VoIP application that is going to be installed on this system. VIPSim may be used by consultants in order to inform different customers about the capabilities of their systems from a VoIP point of view. It may also be used by engineers to troubleshoot existing VoIP systems.

In order to measure the QoS the application needs to measure the values of the parameters that define QoS. They are described in the section 1.1 on page 2. Based on the results given by VIPSim, and compared with some standard accepted values, it is determined whether the system is able to run a VoIP application or not.

VIPSim performs the measurements of the QoS from the end-user (of the system) point of view and not strictly related to the network as someone may assume when talking about data transmission over the network. Because of this, all the factors that influence the QoS need to be considered into these measurements. Based on these factors the components that affect QoS may be identified. Thinking about the sources of delays described in the section 1.1 on page 2, the following components may be identified:

- The VoIP application - affects QoS through the delays it introduces due to its coders operations.
- Operating System (OS) - affects QoS through the delays it introduces due to its procedures of packetization and buffering.
- The Network - may affect the QoS in several ways.
  - through the time it takes for the signal to travel the physical link from one side to the other (traveling delay).
  - through the delays introduced by the analyze of the packets' content performed by the network devices like routers, gatekeepers or firewalls for routing or security reasons.
  - through the bandwidth that is available for communication.

The system that is measured is composed from all the components described above: from the VoIP application and operating system (OS) down to the physical link of the network together with all the network devices involved. In this way there may be seen that all the ISO/OSI layers have an impact on the transmitted data. The OS manages most of these layers, so a great impact on the values of the parameters that affect QoS and especially on the delay, is due to OS' operations. The most important operation of the OS in the performances of the VoIP application, is the OS' scheduler in a multitasking system. The results of the measurements will also reflect this factor.

Almost all the parameters that define QoS in a voice over IP system are related to network packets and not directly to the played sound. These parameters show packets' delay, jitter, lost, out of order, but they do not show anything, explicitly about the sound quality. Although, the sound quality is reflected by these parameters due to the fact that the packets contain the compressed digitized sound data. Therefore, measuring the parameters related to network packets gives a measure of quality of the voice.

In a voice over IP system the packets contain the compressed voice data. The compression and decompression of the sound signal are performed by the coders. The input to a coder is the audio signal sampled, and the output are blocks containing compressed digitized sound given by the coder at a certain rate and with a certain size. These blocks are the payload of the packets that are sent on the network. Measurements of the parameters related to these packets are not influenced by the content of the packets. Therefore, for measurements, these packets do not really need to contain a voice data. It is enough in this case to simulate it, meaning that it is enough to have the same size and creation rate as those provided by coders. So, VIPsim does not contain the implementation of any kind of coders. It will simulate the coders behaviour, by creating network packets with a specified size and at a certain rate. In this way, the coder delay is not considered in these measurements, and the payload of the network packets is not a compressed sound but some dummy data.

VIPSim does not only measure the parameters related to the network packets, but also some parameters that characterize an H.323 terminal. Therefore, the application is implemented based on H.323 recommendation. Although it does not contain a full implementation of an H.323 terminal. The section 1.3.2 will give more explanations on this fact.

1.3.2 Why use H.323?

This section explains why VIPSim needs to be implemented based on recommendation H.323 and why it needs to comply with parts from this recommendation.

Besides the parameters that are related to the network packets, VIPSim is able to give some reports about the H.323 calls. These reports contain information like number of calls established correctly, number of calls that failed to establish and time needed to establish a call. In order to be able to do this, VIPSim needs to be able to use the H.225.0 and H.245 procedures involved in establishing a call between two H.323 terminals.

One of the Gatekeeper's functionality is to provide address/alias translation (almost like a DNS). In order to use this functionality VIPSim needs to be able to communicate with the gatekeeper, to discover and register at it. Therefore it needs to implement the H.225.0 messages that are used for discovery and registration at the gatekeeper, admission to the packet-based network, location translation, and many others described in appendix A.1.

Another feature of a Gatekeeper is the possibility of routing calls through itself, meaning that all the messages and voice data between two terminals are not passed directly from one to another but through the gatekeeper. This may be the case when the H.323 terminals cannot see each other directly on the network. In this way VIPSim also measures the impact of the gatekeeper on the voice quality, which is the delay that this routing may add.

As may be seen from the previous description VIPSim does not need to have implemented all the procedures described in H.323 recommendation. For example, as described in section 1.3.1 it does not need to have the implementation of audio or video coders, because it does not need real voice data in order to perform the measurements. On the other hand VIPSim does not need to be involved in a multipoint conference, so all the functionality of an MCU, MC or MP is not implemented. Yet it will be able to perform multiple simultaneous calls to different destinations.

The components that are implemented in VIPSim are drawn with bold boxes in the figure 1.1. They are the Control System and H.225.0 layer.

Chapter 2

VIPSim Description

This section describes VIPSim's architecture, and shows how and what VIPSim is measuring. This is intended for someone who is going to use VIPSim.

2.1 VIPSim Architecture

As described in section 1.3.1 the application is meant to measure the network packets activity by measuring parameters such as delays, jitter, e.t.c. In order to do this, there is needed one part that produces and sends packets on the network, and one part that receives the packets and computes the measured parameters. In other words there is needed a sender and a receiver for these packets.

One very important measured parameter is the packets' delay, the difference between receiving and sending timestamp of a packet. In order to compute this, the receiver needs the information regarding the time when the packet was sent and when the packet was received. The received timestamp is easy to be retrieved from the system at the receiver, but the sending time needs to be transported along with the packet. Therefore, making the packet to carry along the sending timestamp within its data, makes it possible for the receiver to compute the time needed for the packet to travel, which is the delay between sending and receiving.

The sending and receiving timestamp are given by reading the internal clock value of the system on which the sender and respectively the receiver run. They both run on different machines (systems) that have their own internal clocks that most likely are not synchronized between them. In this way the computation of the delay, which is the difference between receiving timestamp and sending timestamp does not give an accurate result, i.e. does not represent the real value. The solution for this problem is to implement the sender to be the receiver at the same time and in this way they both have the same internal clock to read the time from. Because the packets still need to travel between two points situated somewhere on the network and because there is a sender-receiver application at one side, there must be introduced a reflector that runs on the other side of the link. The reflector's functionality is to send back to the sender the packets it receives. For simplicity the sender-receiver will be called the "Sender" and the reflector will be called the "Reflector", they both being parts of VIPSim. See the diagram 2.1 that illustrates this.

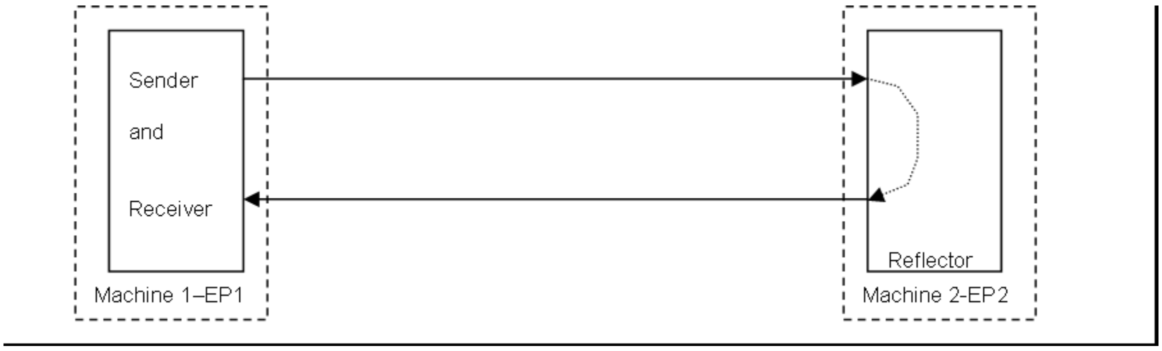


Figure 2.1: Sender - Reflector architecture.

As already presented in the section 1.3.2, VIPSim is implemented based on H.323 recommendation. This means that both the Sender and the Reflector are implemented based on this recommendation. Call establishments between them are performed accordingly to H.323 procedures.

The call establishment procedures are actually an exchange of messages between the two parts. Accordingly to H.323 these messages contain certain information, coded using ASN.1 syntax. H.323 messages are divided into two parts detailed in two other recommendations, H.225.0 and H.245. The description of H.225.0 and H.245 messages is presented in appendix A.1. The figure 2.2 shows the exchange of the H.225.0 and H.245 messages between two endpoints EP1 and EP2 in order to establish a call. It is considered that EP1 performs a call to EP2.

The messages that are exchanged are divided into two stages. First there is the H.225.0 messages stage followed by the H.245 messages stage. When needed, the two endpoints exchange messages (H.225.0/RAS messages) with the gatekeeper (GK) too, for registration and network access.

In order to end a call, terminals have to perform the following procedure.

- EP1 sends "Close Logical Channels" message to EP2 to stop multimedia communication.
- EP1 sends "End Session" message to EP2 to stop the H.245 session.
- EP1 sends "Release Complete" message to EP2 to stop the H.225.0 session.
- EP1 sends DRQ to GK to stop the admission to the network.

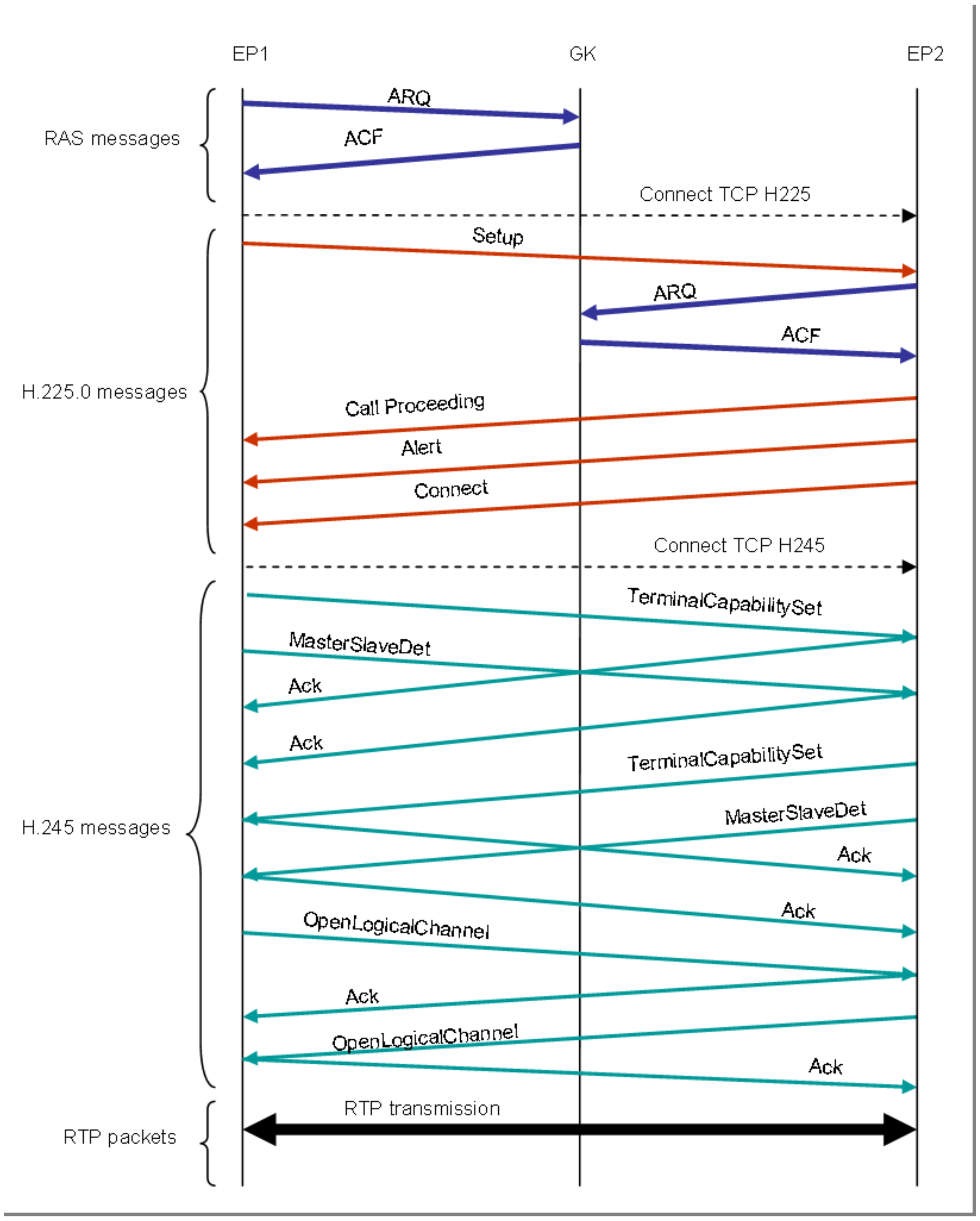


Figure 2.2: Exchange of H.225.0 and H.245 messages for a call establishment.

- GK replies with DCF to EP1.
- EP2 sends "Close Logical Channels" message to EP1 to stop multimedia communication.
- EP2 sends "End Session" message to EP1 to stop the H.245 session.
- EP2 sends "Release Complete" message to EP1 to stop the H.225.0 session.
- EP2 sends DRQ to GK to stop the admission to the network.

- GK replies with DCF to EP2.

If a call is ended either by closing sockets or by disconnections without any receive of the "Release Complete" message in advance, the call is considered as being abnormally terminated (disconnected, aborted, e.t.c.).

Beside the possibility of communication with each other, both the Sender and the Reflector are able to communicate with other H.323 compatible devices due to the fact that they have implemented the H.225.0 and H.245 procedures. These devices are not necessaryly other PCs running H.323 applications or IP phones. They can also be analogue telephones if certain format translation devices such as Gateways are used. A general voice network may look like in the figure 2.3.

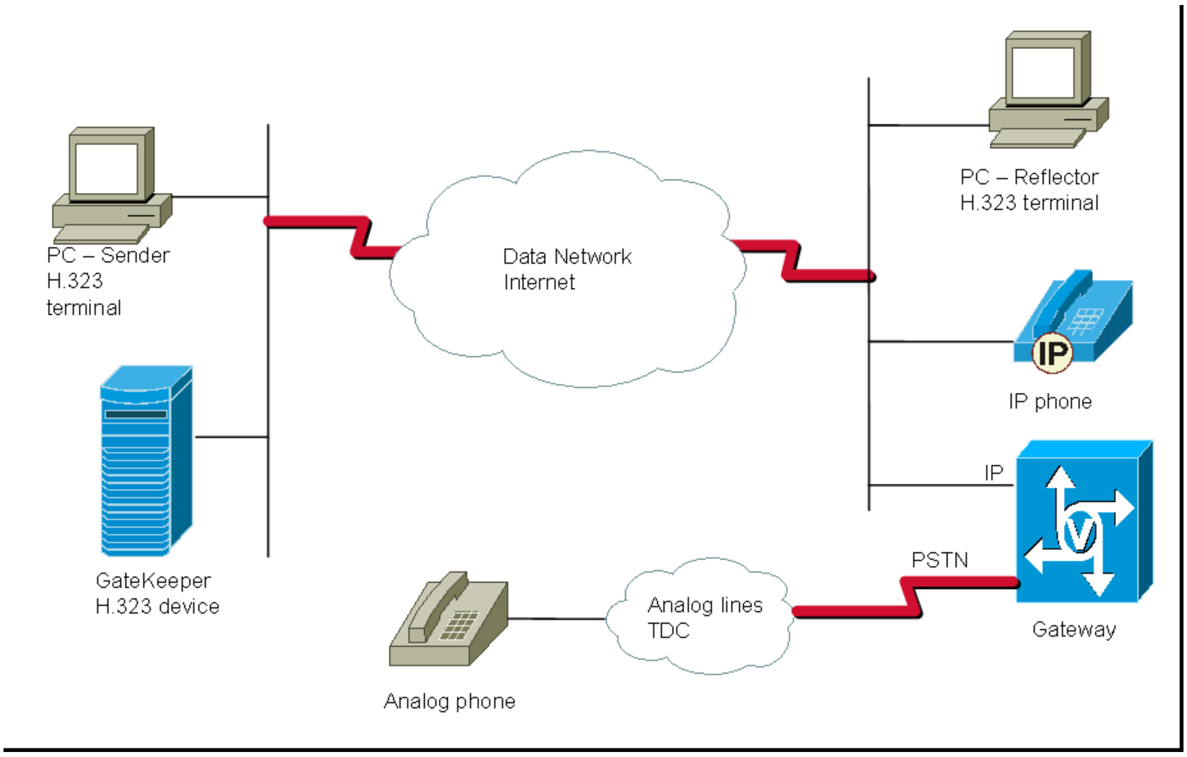


Figure 2.3: A general H.323 network.

If a Sender establishes a voice connection with a different H.323 terminal than the Reflector, the only information that can be measured is the number of calls that were established correctly or failed, the number of normal or abnormal call endings and the time needed to establish the call. The information about packets' delay is not available because any other H.323 terminal different from the Reflector is not able to send back to the Sender, the information contained in the received packet, which is the information that is used by the sender to compute the delay.

On the other hand any H.323 terminal, not only the Sender is able to establish a call to the Reflector. In this case no measurements are performed, but the caller is able to receive back what it sends, so he will hear back his own voice.

One very important feature of VIPSim is that the Sender is able to communicate with several Reflectors, meaning it can establish calls with several Reflectors at the same time. It sends packets to each Reflector and receives them back from each one of them. Each call is a separate call so this configuration may not be compared with a conference (multipoint call). On the other hand a Sender is able to establish multiple calls at the same time with only one Reflector. If there are many Reflectors, the Sender is able to establish multiple calls with each one of them. In the case of multiple calls either between one Sender and one Reflector as well as between one Sender and different Reflectors there are performed call establishment procedures for each call separately and there are opened two logical data channels per call (for both ways of communication). Each call uses its own bandwidth.

2.2 Statistics and Thresholds

2.2.1 Measurements

The measurements performed by VIPSim may be divided into two categories, one category that is related to call procedures and a second category that is related to voice quality. They shall be called "call procedure" measurements and respectively "voice quality" measurements.

"Call procedure" measurements

When a company is implementing a VoIP system, they are transporting a vital service for their business across the network, maybe even the Internet. This means that when the service fails, their users, customers, and the management will immediately notice the event. Because of this, the company needs a tool that can predict bottleneck problems, so they can upgrade their VoIP installation, before the users are affected by potential bottlenecks. Bottlenecks problems are congestions in the flow of information. The gatekeeper is such a potential bottleneck due to the fact that it is involved in each call providing services to H.323 endpoints.

Measuring call setup time, calls established and calls failed as well as the response time of the gatekeeper to the endpoints requests, gives an indication of what the user experiences when dialling a number trying to establish a call. Simulating multiple calls can be used to stress test the gatekeeper, giving important information about the overall capacity of the VoIP network from a gatekeeper's perspective. Also when designing, and installing a VoIP solution from a consultant's perspective, the "Call Procedure" measurements performed by VIPSim are used to assure that the gatekeeper is working as expected. Measuring the gatekeeper service response time, for all requests, may be used to identify upcoming performance issues.

"Call procedure" measurements provide information about the following parameters:

Call Initiation Counters. The call establishment procedure starts with the "Setup" message and ends when logical channels for data transmission have been successfully opened (see section 2.1). In this moment the call is considered established successfully. If one of the H.225.0 or H.245 message exchange fails the call establishment fails. The Sender keeps a counter that counts the number of calls established successfully and a counter that counts the number of calls that failed.

Call Termination counters. The termination procedure of a call is initiated by sending or receiving a "Release Complete" message. The termination of a call without the receiving of such a message in advanced is considered as an abnormally call termination. The Sender has a counter that counts the number of calls terminated correctly and a counter that counts the number of calls terminated abnormally.

Response Time of Gatekeeper. The gatekeeper replies with an answer to each request received from an endpoint. The Sender keeps the record of the time it took to receive the reply. This time is the response time of the gatekeeper.

Call Establishment Time. The Sender also measures the time needed to establish a call, since the "Setup" message was sent, until the last channel was opened successfully, i.e. the last H.245 message (Open Channel Acknowledge) was received.

"Voice Quality" measurements

As mentioned in the section 1.3.1 in order to measure the sound quality, VIPSim measures the parameters that defines QoS, and these parameters give an indirect information about the sound quality.

The measured parameters related to sound quality are listed below. They are defined in section 1.1 on page 2.

Delay (Latency) - the time difference between the receiving and sending of a packet. Its a round trip delay (from the Sender to Reflector and back).

Jitter - the variance of the interarrival time.

Interarrival time - the delay between received packets.

Packets Lost - packets sent that do not arrive back.

Packets Out of Order - packets that are not received in the order they were sent. The information contained within their payload is discarded.

An excessive delay will cause two problems: echo and talker overlap if they exceed certain values (see section 1.1). Measuring the delay during a certain period of time may give information of when these problems may appear, how often and how severe, and in this way the user is able to upgrade its VoIP system in order to prevent this and to provide better quality to his customers.

The client playing the voice sample needs to be able to play it at a fixed rate. An excessive jitter may cause interruptions of the played sound because it is not able to play it at that certain fixed rate. In order to compensate for jitter, the client has to implement a receiving buffer that accumulates several voice data packets before releasing them to the player. Measuring jitter, gives an important information about how big receiving buffers the client should have.

The interarrival time is not one of the parameters listed in section 1.1 where the QoS is defined but it is strongly related to the jitter (see the definition in the mentioned section). While the jitter is a number showing the spread of the delay of packets, the interarrival time shows the rate the packets arrive with. It is helpful in order to decide about the sending rate and packets size, or, in other words, about the coder to use at the source. On the other hand it may help to detect bottleneck problems and to localize them in time in order to correlate them with some other events. Finally this might help to localize the source of the problem.

Measuring lost packets and packets out of order, is important. Each packet holds voice data for a period of time depending on the coder. Losing or discarding a voice data packet, means the client has to guess how the signal would look like. In the human ear words would be chopped, and if packet loss exceeds 5-10% it is difficult for the human brain to recognize the words of the conversation.

The gatekeeper has a big influence over these results when the traffic is routed through it, meaning that the gatekeeper routes the audio/video data exchanged between two endpoints through itself. Transferring traffic through the gatekeeper is probably the most important test, also when comparing the result with a direct call. The delay introduced by the gatekeeper can be measured in this way and together with the results obtained from the "Call Procedure" measurements may establish the gatekeeper's performances.

Both "Call Procedures" and "Voice Quality" measurements are performed for certain time intervals defined by the user. These time intervals are called sessions of measurements, and the gathered data within one session is saved into a data-base. A session of measurements may range from a few minutes to a few days.

2.2.2 Thresholds

One very interesting feature of VIPSim is that it may be converted from a measuring tool that only records measured values into a monitoring tool that gives warnings and performs actions when certain events occur. These events take place when the measured values touch certain fixed values defined by the user. On one hand this is useful when the user needs to be informed about the current state of the performance of the system in order to be able to intervene manually to increase the performances if necessary. On the other hand this is useful when the user desires to make the application to perform certain actions upon the occurrence of these events, again in order to increase the performances of the system.

These fixed values that the measured values are compared against are called threshold values. The user defines them for each measured parameter separately. The user also defines the type of action he wants VIPSim to perform. As described above these actions are divided into two categories:

- Warnings received by the user in form of visual/audio alarms or written notices like emails, SMS or log files.
- Operations performed automatically by VIPSim.

These two types of action are implemented in VIPSim simply by providing an application name that is executed by VIPSim when the events occur and the necessary arguments needed to be passed to it. This application will execute further the desired operation (warning or action).

There may be two threshold values: higher threshold and lower threshold. The lower threshold value is lower than the higher threshold value. The range of value between these two threshold values is called the normal value range that the parameter may take. It is not necessary to have both threshold values defined, as it is not necessary to define them for all the measured parameters. If one value is not defined than the other defined threshold value is the border between the normal value range and the abnormal value range. The current measured value is compared against the higher and/or lower threshold value to check if it is inside or outside its normal range. If it is outside this range a "threshold reached" or "threshold exceeded" event occurs that triggers the defined action to be performed.

If a threshold value is reached or exceeded an action is performed as described above. When the measured value goes back to normal, meaning that is goes below the higher threshold and above the lower threshold, the mechanism is armed and is ready to trigger again the action. More complex rearming conditions may be defined. For example a mechanism may rearm only if the current measured value goes back to normal range after it was above the higher threshold (or below the lower threshold) several times in a row.

2.2.3 Presenting the Results

All the results obtained from the measurements performed by VIPSim may be presented in tables. Below there are described tables resulted from both "Call Procedures" and "Voice Quality" measurements. These tables are created per session of measurements.

The first table contains general information about each call that exists in a session of measurements. There will be such a table for each call. This general information is taken from the user input or determined by the application but it is not the result of the measurements. The information may be seen in the table 2.1.

Name	**MU**	**Type**	**Description**	**Example**
CRV	-	integer	Call Reference Value. Identifies the call.	32001
Source IP	IP	IP	Source of call - IP address	192.168.1.1
Destination IP	IP	IP	Destination of call - IP address	192.168.1.2
Bandwidth	Kbit / s	integer	Bandwidth used by this call	64
Packet size	Bytes	integer	The size of the packets payload	160
Packet rate	ms (fps)	integer	Packet sending rate	40 (25)

Table 2.1: General information about a call (Input data, not measured).

The Call Reference Value is an integer value between 1 and 65535. The packet size is the size of the packets payload, without the header (see section 2.3.1 for details about packets header). The packet rate is the rate of creation of packets with data that are sent on the network. The rate may be expressed either by giving the time interval between packets in milliseconds (ms) or by giving the number of frames (packets) that are created per second (fps).

$ms = \frac{1}{fps}$

The next two tables contain information related to "Call Procedure" measurements. The table 2.2 contains information about a single call. In one session of measurements there will be such a table for each existing call. The table 2.3 contains overall information about all the calls that exist in a session of measurements.

Name	**MU**	**Type**	**Description**	**Example**
Call established	true/false	Boolean	Call was established correctly or it failed.	true
Call ended OK	true/false	Boolean	Calls ended correctly or abnormally.	true
Call establish time	ms	integer	Time needed to establish the call.	125

Table 2.2: Call related measured data for a single call.

Name	UM	Type	Description	Example
Total calls	-	integer	Total number of calls.	10
Calls established	-	integer	Calls established correctly.	10
Calls failed	-	integer	Calls not established due to a failure.	0
Calls ended OK	-	integer	Calls ended correctly.	10
Calls disconnected	-	integer	Calls ended abnormally (disconnected).	0

Table 2.3: Call related measured data for all calls.

As mentioned in section 2.2.1 on page 14 during the "Voice Quality" measurements, parameters that are related to the packets of data that travel between two terminals are measured. These parameters are the packets delay, jitter, interarrival time, packets lost and out of order. During a session of measurements there are many packets that travel between the source and the destination, so the amount of data obtained by recording the specified information about each packet, creates huge data that needs to be stored. In order to save space this huge amount of data is reduced by using mathematical formulas to a smaller data structure called **Statistic**. The mathematical formulas used, reduce the amount of data and they preserve their meaning. The following kinds of computations are made:

- The delay of all packets are used to compute minimum value of the delay, mean (average) value of the delay and the maximum value of the delay.
- The interarrival time of all packets are used to compute minimum value of the interarrival time, mean (average) value of the interarrival time and the maximum value of the interarrival time.
- From the sending rate and the interarrival time one value is computed (see section 2.3.2) that is the jitter.
- The number of packets received are counted
- The number of lost packets are counted
- The number of packets that arrive out of order are counted

These computes values are the values form a Statistic. Not the whole data gathered in one session of measurements is reduced to one Statistic. The session of measurements is divided in time intervals and Statistics are computed for each such a time interval. The results may then be presented in a table where each column represents a parameter from the data structure and each row represents a certain Statistic created for a certain time interval. For example consider a session of measurement of 24 h with one call and that the time interval is one hour. So the table will have 24 rows and the time column will show a saving time every hour. This time interval is defined by the user before the start of measuring session. The structure of such a table may be seen in table 2.4 where the second row contains the description of each column instead of measured data.

Based on the data from the table 2.4 graphical representations may be created showing how values of the delay, the interarrival time, packets lost or out of order changes

Time	**CRV** call reference value	**Received packets**	**Lost packets**	**Packets out of order**
date and time when the Statistic was saved	CRV to identify the call the packets belong to	total number of packets sent	packets that were not received back	packets not received in the same order they were sent

Jitter	**minimum delay**	**mean delay**	**maximum delay**
The Jitter computed value	...	...	...

minimum interarrival time	**mean interarrival time**	**maximum interarrival time**
...	...	...

Table 2.4: Packets related measured data.

over the time. Then these graphics may be compared to observe differences between sessions with one call and sessions with multiple calls to find the limitation of the system, or to find out the moments when bottleneck problems occur to make it easier to detect their sources.

If some measured parameters take values that are not in a normal or acceptable range, it is interesting to know how many packets generated these abnormal values, in order to see if it was an isolated problem or a continuous one. For example, if the maximum delay shows a value greater than the acceptable value of 150 ms it is interesting to know if there are many packets with this delay or just one that determined such a maximum value. The answer to this question may be given by creating histograms. Histograms are graphical representation of data that is classified into groups characterized by a range of values. VIPSim has implemented an algorithm that creates a data structure that may be used to draw histograms of delay and interarrival time.

In order to create this, all the received packets are distributed into groups that represent a certain time interval based on their delay or interarrival time. For example group 1 may represent all time intervals with values between 0 - 49 ms, group 2 may be between 50 - 99 ms, and so on. There should be a finite number of groups, so the last group is considered to represent all the time intervals greater than a certain value. There will be one histogram for delays and a different one for interarrival time. For each received packet of data the delay is computed and the packet is then placed into the group the delay belongs to. As example, if the delay is 2.4 ms the packet belongs to group 1 (0 - 49 ms). During a session of measurements with each packet received back, the number of packets in different groups increases, some of them more than others.

Histograms are graphical representation of the data created as described above. On the horizontal axes the groups representing the time intervals are shown, and on the vertical axes either the number of packets (in groups) or the percentage (number of packets in a group relative to the total number of packets). Such a histogram may look like in the figure 2.4.

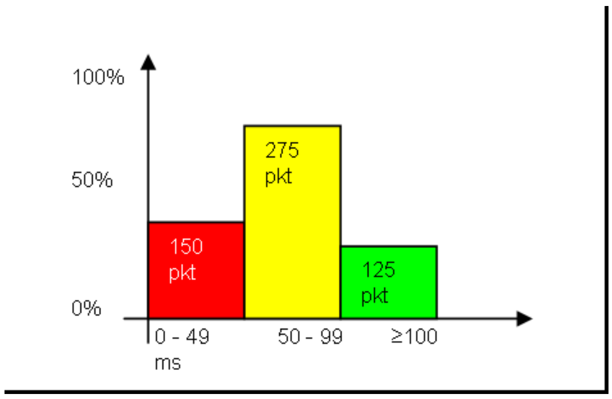


Figure 2.4: Packets delay histogram.

These histograms are computed for the whole session of measurements and not only for one time interval like Statistics. The precision of the histograms, which is the range of values of each group, is defined by the user before the start of the session of measurements. The same goes for the limit of the histogram, which is the value that defines the last group. All the packets that have the delay greater than this value go into this last group. In the figure 2.4 this value is 100 ms.

2.3 Measurements mechanism

2.3.1 Packet description

A VoIP application uses network packets containing voice data in order to transmit audio signal between two terminals. In the same way VIPSim uses these packets in order to measure the quality of the voice. As mentioned in section 1.3.1 these packets do not contain the real voice data like a normal VoIP application but some dummy data. The general format of the packets is the same like in an ordinary VoIP application, in order to enable VIPSim to communicate with other H.323 terminals as mentioned in section 2.1.

Like in all multimedia transmissions that take place on Internet these packets have to comply with certain protocols and formats. Each packet format imposed by protocols adds a header of information to the packet's data like a layer. Due to the

IP header - 20 bytes
UDP header - 8 bytes
RTP header - 12 bytes
RTP payload data

Figure 2.5: RTP packet format.

fact that these packets belong to a multimedia transmission, they have the format imposed by the Real-Time Protocol (RTP). These packets are sent on non-reliable connections over the Internet, so the packets have further UDP and than IP layers. Therefore besides from the payload (the actually voice data) each network packet has some extra information regarding each of these layers (RTP, UDP, IP), attached to it as three headers.

The exact format of the header of UDP and IP is not interesting for the purpose of the description made in this chapter, but the header format of RTP protocol is important and it will be described later. An important thing to know is that the total size of these three headers together is 40 bytes, so actually the total size of a packet sent on the network is the size of the payload data plus these 40 bytes. The packet format looks like in the figure 2.5.

In order to compute the time needed for a packet to travel from source to the destination (packet delay), as well as the jitter and interarrival time, the destination needs the timestamp when the packet was sent and when the packet is received. The timestamp of sending time needs to travel along with the packet, so it should be present in the packet content. Otherwise it is impossible to know the packet's sending time upon receiving it. On the other hand in order to find out if a packet does not arrive (it is lost) or if a packet arrives in the wrong order they need to have a sequence number. Therefore this information needs to travel with the packet too. So the packets must contain within their data a sequence number and the sending timestamp.

There may be seen that the sending timestamp and the sequence number are present in the RTP header. The RTP header looks like in the figure 2.6.

The sequence number of a packet is a 16 bits value (0 - 65535) and the timestamp is a 32 bits value.

Because in these measurements a very good precision is required, the system's functions that retrieve the time with the highest resolution is used as described in section 4.1. The "Performance counter" gives the time value with the highest resolution. This counter's value increases several millions of time per second, meaning that it has a resolution less than 1 microsecond. The value of this counter is a 64 bits value, and thus it does not fit into the RTP header. In order to solve this a new header

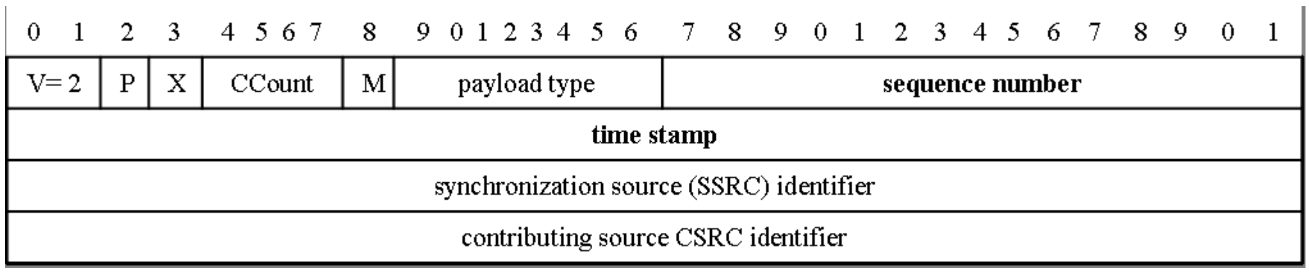


Figure 2.6: RTP packet header format.

is created that contains the 64 bits timestamp value. Unlike the other headers this new header is not attached to the payload data not to change its size, but is created from the first bytes of the payload data. The rest of the payload contains undefined data, in order to fill in up to the necessary size of the packet. The format of the new packet, reffered as VIPSim packet looks like in the figure 2.7.

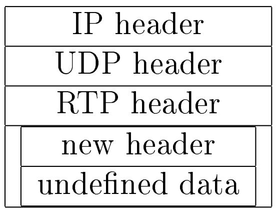


Figure 2.7: VIPSim packet format.

Before the packet content is delivered to the OS in order to be sent on the network, both the RTP header and the new header are filled with necessary information (sequence number, timestamps, e.t.c.). The timestamp is obtained by querying the performance counter value just before putting it into the packet content. Upon receiving the packet the performance counter value is queried again to obtain the received timestamp, and next the computation algorithm is performed on this data to obtain the results.

2.3.2 The Algorithm Used to Compute the Results

The results recorded by VIPSim are not always directly obtained from the measurements. Some computations are required on the measured values to obtain the desired parameters.

For the "Call Procedure" measurements there is no need for a special algorithm to compute the parameters. Number of calls with certain characteristics are just counted. The time interval in which a call is established is again easy to compute by making the difference between absolute time at the end of call establishment procedure and the absolute time at the start of the procedure. The time is then expressed in milliseconds. The same goes for the response time of the gatekeeper.

For the "Voice Quality" measurements, due to the fact that the huge amount of data obtained is reduced to a few values, as mentioned in section 2.2.3, a special algorithm has to be developed. This algorithm also contains the creation of the data structure (updating the groups) used for the histograms.

The data gathered during a session of measurements is divided into groups, each group covering a certain time interval, and such a group of data is reduced to a single data structure that is a Statistic. So, a session of measurements contains as many Statistics as time intervals the user divided it into. A Statistic contains the measured data as mentioned in section 2.2.3 plus the time when the Statistic is saved and the CRV (call reference value) identifying the call as may be seen in the next list:

- saving time
- CRV
- packets delay (Latency)
- packets interarrival time
- jitter
- number of packets received
- number of packets lost
- number of packets out of order

In order to compute this, a Statistic needs to keep some other auxiliary variables, like total delays or total interarrival time, first and last received packet number, that will be described later. Therefore the Statistic data structure is defined as below:

```
typedef struct {
        int number_packets;
        double total_delay, total_jitter;
        double maximum_delay, minimum_delay, mean_delay;
        double maximum_interarrival, minimum_interarrival, mean_interarrival;
        double jitter;
        int out_order_packets;
        int lost_packets;
        int last_received_packet_sequence_nr;
        int first_received_packet_sequence_nr;
        hyper previous_Packet_Receive_Time; //hyper = 64 bits
        hyper previous_Packet_Sending_Time;
        int CRV;
} statistic_t, *pstatistic_t;
```

The total delay and total interarrival time keep the sum of the delay and respectively interarrival time of all received packets, and it is used to compute the mean values. The first and last received packet sequence number keep the sequence number of the first and the last packet that was involved in computation of the respective Statistic. They are used in obtaining the lost and out of order packets. The *previous_Packet_Receive_Time* and *previous_Packet_Sending_Time* variables keep the sending and receiving time of the previous packet, used in computation of current interarrival time and current intersending time that are used further in computation of the jitter.

With each packet that is received the Statistics parameters are updated as follows:

- Upon receiving a packet the difference between the received timestamp and the sent timestamp is computed and the result is kept in a variable called *current_delay.*
- The difference between the received timestamp of the latest packet and the received timestamp of the previous packet is also computed and kept in a variable called *current_interarrival.*
- The difference between the sent timestamp of the latest packet and the sent timestamp of the previous packet is also computed and kept in a variable called *current_intersending.*
- The *total_delay* and *total_interarrival* variables keep, as defined, the total delay and respectively the total interarrival time until the current moment.
- The variable *number_packets* keeps the total number of packets received until the current moment.

Then for each received packet the Statistics parameters are updated using the following algorithm:

$$number\_packets = number\_packets + 1$$

$$total\_delay = total\_delay + current\_delay$$

$$total\_interarrival = total\_interarrival + current\_interarrival$$

$$mean\_delay = \frac{total\_delay}{number\_packets}$$

$$mean\_interarrival = \frac{total\_interarrival}{number\_packets}$$

$$minimum\_delay = \text{Minimum}(minimum\_delay,\ current\_delay)$$

$$maximum\_delay = \text{Maximum}(maximum\_delay,\ current\_delay)$$

$minimum\_interarrival$ = Minimum($minimum\_interarrival, current\_interarrival$)

$maximum\_interarrival$ = Maximum($maximum\_interarrival, current\_interarrival$)

$D = current\_interarrival - current\_intersending$

$jitter = jitter + \frac{|D|-jitter}{16}$

The jitter is calculated accordingly to the formula defined in recommendation H.225.0 [2], see appendix A.5. This formula is an estimation of the statistical variance of the packets interarrival time. The Minimum and Maximum are standard mathematical functions that compute the minimum and maximum value between two input values.

As mentioned in the beginning of this section the session of measurements is divided into time intervals for which a Statistic is computed. This means that a Statistic is updated with a certain number of packets that are received within that time interval. So, there may be considered that a Statistic contains information about packets that have the sequence number within a certain range. This information is saved in the Statistic data structure in two variables called *first_received_packet_sequence_nr* and respectively *last_received_packet_sequence_nr*. When one time interval elapses the Statistic should be closed and saved and a new one should be started. This may introduce some errors in the results because of the following scenario:

After a Statistic is closed a very late packet may arrive that normally would belong to the closed Statistic instead to the newly created one due to the fact that its sequence number is less then the *last_received_packet_sequence_nr* of the closed Statistic and so, it should contribute to the number of packets out of order from the closed Statistic not from the newly created one.

In order to solve this the following solution is used:

When a time interval elapses the current Statistic is not closed but just put into a waiting state, in which it waits for late packets. The newly created Statistic will then be updated only with received packets that have the sequence number greater than *last_received_packet_sequence_nr* of the previous Statistic that is waiting. Only when the newly Statistic is about to end due to the elapsing of the last time interval, the one that was waiting is closed and is put into a third state, that is "ready to save". A statistic in the third state waits to be saved by the application into the database. In this moment the current Statistic goes into the waiting state and a new one is created. So there are defined three states of Statistics:

- active Statistic (or current Statistic)
- waiting Statistic (or previous Statistic)
- ready to save Statistic

The algorithm that is used to perform the correct update of the Statistic using the formulas described in this section is described below. The action flow of this algorithm may be seen in the appendix A.3.

- When a packet arrives the *current_delay* is computed as the difference between received and sent timestamp as well as the *current_interarrival* as the difference between this packet received time and the previous packet received time. Also the *current_intersending* parameter is computed as the difference between sending time of the last packet and the sending time of the previous packet. The total number of received packets is increased by 1 (*number_packets* = *number_packets* + 1).
- There is checked if the packet belongs to the current Statistic or to the previous one, by comparing its sequence number with the sequence number of the last packet from the previous Statistic.
- If it belongs to the previous Statistic there is checked if it is not too old, by comparing its sequence number with the sequence number of the first packet from the previous Statistic. If it is, then the packet is discard, being considered lost (*lost_packets* increases by 1), otherwise it is considered out of order (*out_order_packets* increases by 1).
- If it belongs to the current Statistic there is checked if it is in the right order using its sequence number. If it is not in the right order, it is considered out of order in the current Statistic (*out_order_packets* increases by 1).
- If it is in the right order the following steps are performed:
    - Check for lost packets. If the newly received packet does not have the sequence number equal to the previous received packet sequence number plus 1 then all packets having the sequence numbers between these two values are considered lost until they arrive (becoming out of order).
    - The minimum, mean and maximum delay are computed using the formulas described above. The delay histogram is updated by increasing the number of packets in the right group based on the packet delay.
    - The minimum, mean and maximum interarrival time are computed using the formulas described above. The interarrival time histogram is updated by increasing the number of packets in the right group based on the packet interarrival time.
    - The jitter is computed using the formula described above.

Chapter 3

VIPSim Design

This section shows the internal design of VIPSim describing the classes, the threads and their functions. In this section the reader may also find how the voice packets transmission mechanism works. This section is intended for someone who whant to understand VIPSim's implementation better and maybe to continue its developing.

3.1 Classes and Objects

As it is presented in section 2.1 VIPSim consists of two components, a Sender and a Reflector. Both the Sender and the Reflector are the same application, which is VIPSim. VIPSim may run in one of these two modes, either as a Sender or as a Reflector.

The implementation of VIPSim is based on an Object-Oriented design. Its whole functionality may be seen as a sum of smaller tasks. The application is in this way divided into modules that perform each of these tasks. A module is one or a group of several objects, each one described by a class. These classes define the functionality of the modules and its characteristics. The main modules that the application may be divided into may be seen in the next list:

- endpoint behaviour and characteristics. See table 3.1.
- communication with the gatekeeper (including support for RAS messages). See table 3.2.
- support for H.225.0 - call control messages. See table 3.3.
- support for H.245 - control messages. See table 3.4.
- RTP transmission and statistic computation. See table 3.5.
- packets sending module. See table 3.6.
- statistic saving module (to database). See table 3.7.

The class diagram may be seen in the appendix A.4. This diagram is not a complete class diagram. It does not show all the methods and member variables of the classes.

The classes that describe threads objects are shown by displaying their function rather than the class name with its methods and variables. For a detailed description of all the threads see section 3.2.

Class name	**Description**	**Uses thread**
H323BaseEndpoint	Base class that defines the common methods and characteristics of a general endpoint like endpoint type, endpoint capabilities, endpoint alias addresses (phone number, user name, email).	-
H323LocalEndpoint	Class derived from H323BaseEndpoint defining characteristics and methods for a local endpoint.	-
H323RemoteEndpoint	Class derived from H323BaseEndpoint defining characteristics and methods to handle a remote endpoint.	End Call Incoming Call
EndpointType	Defines the endpoint type (country code, manufacturer code).	-
MyCapabilitySet	Defines the audio/video or data transmission capabilities of the endpoint.	-
Q932Display	Defines the display name, information that goes into Q931 packets.	-
Q931BearerCap	Defining the request for a bearer that needs to be provided by the network [2].	-
LogicalChannel	Defines the communications channels for both directions. It keeps information about the type of data that is going to be transmitted on these channels as well as the channels addresses (IP and port numbers).	-
End Call thread	Thread used in performing the call termination procedure.	-
Incoming Call thread	Thread used in performing the call establishment procedure for a called endpoint.	-

Table 3.1: H.323 Endpoint module classes description.

In a call there is always a source and a destination. Both of them are represented by an instance of VIPSim running on two different machines involved in the call. On the machine that is the source of the call, VIPSim runs in the Sender mode, and on the machine that is the destination of the call, VIPSim runs in the Reflector mode. Both instances of VIPSim keep information about the source endpoint and about the destination endpoint. This information is kept in *H323LocalEndpoint* and *H323RemoteEndpoint* objects. Depending on the mode VIPSim runs in, these objects represent the source or the destination of the call. For example if the instance runs as a Sender then, the local endpoint is the source of the call and the remote endpoint is the destination of the call, and vice-versa.

One instance of VIPSim always contains one instance of the object *H323LocalEndpoint* representing the local endpoint but it may contain more than one object *H323RemoteEndpoint* depending on how many remote endpoints it is in a call with.

The *H323LocalEndpoint* also defines the interface with the user, providing him with functions for accessing the gatekeeper and managing the calls.

Class name	**Description**	**Uses thread**
Gatekeeper	Provides functions for performing requests to the gatekeeper like: discovery, registration, admission on the network, location translation, bandwidth change, unregistration and disengage.	-
RASControl	Provides encoding and decoding as well as managing of content of RAS messages.	Receive RAS
Receive RAS thread	This thread receives RAS messages from the gatekeeper.	-

Table 3.2: Gatekeeper communication module classes description.

Class name	**Description**	**Uses thread**
H225CallControl	Provides encoding and decoding as well as managing of content of Q931/H.225.0 messages.	Receive H2250
H225Listener	Listens on a server socket for H.225.0 connections from clients that are calling endpoints.	Listener H2250
Receive H2250 thread	Receives the Q.931/H.225.0 messages.	-
Listener H2250 thread	Listens and accepts client connections.	-

Table 3.3: H.225.0 support module classes description.

Class name	**Description**	**Uses thread**
H245Control	Provides encoding and decoding as well as managing of content of H.245 messages. It also provides listener functionality on a server socket for H.245 connections.	Receive H245 Listener H245
MSDSE	Defines handling methods for Master Slave Determination procedure (See appendix A.2).	-
Receive H245 thread	Receives the H.245 messages.	-
Listener H245 thread	Listens and accepts client connections.	-

Table 3.4: H.245 support module classes description.

The *InfoPacket* object was introduced in order to pass information regarding the packets (sequence number, sending, receiving time) between receiving and computing functions.

Class name	**Description**	**Uses thread**
RTPUDPSession	Defines a RTP session, which is a single call with 2 channels. Handles opening of RTP channels and manages the content,transmission and reception of data packets (RTP payload).	Receive RTP Receive RTCP
InfoPacket	Defines the content of data packets (timestamps, sequence number).	-
Statistics	Defines Statistic and histograms data structures and handles the computation of measured parameters and of the Statistic.	Compute Statistic Place Data
StatisticPointer	Defines a pointer to a Statistics object.	
Receive RTP thread	Receives RTP packets and place their information in a queue for computations.	-
Receive RTCP thread	Receives RTCP packets.	-
Place Data thread	This thread place the "ready to save" Statistic in the saving queue.	-

Table 3.5: RTP transmissions and Statistic computation module classes description.

Class name	**Description**	**Uses thread**
SendManager	Manges the sending of RTP packets at a certain fixed rate.	Sending
Sending thread	Sends the packets on the network.	-

Table 3.6: Packets Sending module classes description.

The "Sending" thread contains a loop that continuously reads the time in order to determine the moment it has to send the packets on the network (see section 3.3 for details). It sends packets from each RTP session, which means for each established call, at a certain fixed rate defined by the user.

Class name	**Description**	**Uses thread**
DBHandler	Manages the database connection and saving of the "ready to save" Statistics placed in the saving queue.	Saving
Saving thread	Saves actually the data into data-base.	-
PCBuffer	Defines a Producer/Consumer buffer (queue) for the "ready to save" Statistics.	-

Table 3.7: Data-base Saving module classes description.

When a *Statistics* object has a Statistic that is "ready to save" it puts itself in a queue through the "Place Data" thread. The *DBHandler* object reads the *Statistics* objects from this queue and saves the Statistic data structures contained within them into the data-base through the "Saving" thread. This queue is defined by the *PCBuffer* object. *Statistics* written italic refers to the object and Statistic refers to the data structure that keeps measured (computed) parameters defined in section 2.2.3.

The way all these objects and threads work together to establish a call and to start a measurement session is described below. Between parentheses there are also mentioned the modules that handle the described operations.

- The endpoint that waits for H.323 calls, which is the destination of the call or the Reflector, register at the gatekeeper if necessary (gatekeeper module) and then opens an H.225.0 server waiting for connections (H.225.0 module).
- The caller endpoint, which is the source of the call or the Sender, creates a remote endpoint object (endpoint module) then register at the gatekeeper and requests admission for a call (gatekeeper module) if necessary and then establishes a TCP connection to the destination, which is the created endpoint.
- Upon connection of the client, the receiver also creates a remote endpoint (endpoint module), and the H.225.0 and H.245 messages exchange starts (H.225.0 and H.245 modules together with endpoints modules from both sides).
- When the call is established, the packets transmission and reception is started and the measurements are performed (RTP transmission and Statistic computation module).
- The Sending module is used here to send the data packets at a certain rate, and the Saving module is used to save the computed Statistics into the data-base.

3.2 Threads description

VIPSim is a multi-threaded application. Each operation that needs to be performed in parallel with others is implemented in a separate thread. There are general threads, common for the whole application, and there are threads created for each separate call.

Common threads and their functionality is described below:

Receive RAS thread - This thread contains a reading operation from a socket. The socket is created for an unreliable data transmission between the endpoint and the gatekeeper. The reading operation is blocking, meaning it waits for the data to be available in the receiving buffer, blocking the thread execution until a RAS message is received. The reading operation is implemented in a thread in order to let the rest of the application execute while the thread is blocked waiting to receive data.

H.225.0 Listener thread - This thread contains a listen operation on a socket. This socket is a server socket that receives H.225 client connections, meaning it receives H.323 calls. The listen operation is blocking meaning it waits for the client to connect. This is the reason it is implemented in a thread, in order to let the rest of the application run while it waits for connections.

Sending thread - This thread implements the mechanism that sends data packets with a certain rate (see section 3.3).

Saving thread - This thread implements the saving procedure of Statistic data into the data-base. It was introduced in order to have a single object that handles the database operations, not several threads (due to multiple calls) that tries to access the database at the same time.

Next the threads that are created at the same time with the creation of a remote endpoint are described. The operations implemented in these threads are used for controlling the call and for performing the measurements between the local endpoint and the remote endpoint. For each single call these threads are created.

Receive H.225.0 messages thread - This thread contains a reading operation from a socket like the "Receive RAS" thread described above. The difference is that this socket is created for H.225.0 communication that is made on a reliable connection.

H245 Listener thread - This thread contains a listen operation on a server socket like the "H.225.0 Listener" thread, but the connection is established for H.245 communications between the two endpoints.

Receive H.245 messages thread - This thread contains a reading operation like "Receive H.225.0 messages" thread but for H.245 messages.

Incoming Call thread - This thread is created when an incoming call is signalled, and it implements the call establishment procedure from the called endpoint's point of view. The caller does not implement the call establishment procedure in a thread, because it does not perform several call establishment procedures in parallel.

Make Call function - main thread - This operation starts and performs the call establishment procedure from the caller's point of view. This operation is not shown as a separate thread in the class diagram because it is a function that runs in the main thread of the application.

End Call thread - This thread handles the call termination procedures.

Receive RTP thread - This thread contains a blocking receive operation like the ones described above but for RTP packets.

Receive RTCP thread - The same like "Receive RTP packets" thread but for RTCP packets.

Place Data thread - This thread place the computed Statistic data that is ready to be saved into the saving buffer from where the saving module will read it and save it into the data-base.

These threads may be divided into two groups. One that handles the call procedures (establishment and termination of a call) and a second group that handles the content and transmission/reception of RTP packets and the computation of the Statistics from the measured parameters.

Call Procedures threads

The Call Procedure threads and where they are created is described below:

- Receive of RAS messages thread - one instance running on the Sender and one instance running on the Reflector.
- H.225.0 Listen thread - only one instance running on the Reflector.
- H.225.0 Receive thread - one instance running on the Sender and one instance running on the Reflector.
- H.245 Listener thread - only one instance running on the Reflector.
- H.245 Receive thread - one instance running on the Sender and one instance running on the Reflector.
- Incoming Call thread - only one instance running on the Reflector.
- Make Call function (main thread) - this function is called only on the Sender.
- End Call thread - one instance running on the Sender and one instance running on the Reflector.

The appendix A.6 shows using Petri-Nets the action flow of these threads as well as the communication between them. The action flow shown in the diagram is described below:

The call establishment starts with the Make Call function that is called on the caller (the Sender). This will determine on the Reflector, due to a sent message (Setup message), the start of the Incoming Call thread. As it may be seen from the diagram, the Make Call function communicates with the Incoming Call thread by means of messages that are sent and then received through the H.225.0 and H.245 threads. For all the messages that require reply timers are implemented in Make Call function and Incoming Call thread. If these timers time out the call establishment procedure ends.

The H.225.0 and H.245 threads receive messages sent by the other side and generate events related to these messages. These events determine the Make Call function and Incoming Call thread to continue that means to send the next H.225.0 or H.245 message. The H.225.0 thread from the Reflector is the one that starts the Incoming Call thread upon receiving of Setup message. It may also start/resume the End Call thread upon receiving of a Release Complete message. The H.245 thread also send a replies to messages like Terminal Capability Set or Master Slave Determination.

The call termination procedure is performed by the End Call thread that is started by the call of HangUp function from the Sender, and by the Reflector due to reception of the Release Complete message. This thread sends a Release Complete message as reply if necessary and then closes the RTP, H.225.0 and H.245 connections for both sides.

RTP packets and Statistic computation threads

The RTP packets transmission and Statistic computation threads are:

- Sending thread - only one instance running on the Sender
- Receive RTP thread - one instance running on the Sender and one instance running on the Reflector
- Place Data Receive thread - only one instance running on the Sender
- Saving thread - only one instance running on the Sender

The diagram from appendix A.8 shows how information travels from one thread to another. The Send thread sends the data through the network packets to the Reflector that receives it and sends it back to the Sender. Here the Receiving thread reads the data from the network packet and it passes it to the Computation function. This function, using the received packet data updates the Statistic structure. When the Sending thread finishes the sending of a certain number of packets, it resumes the thread that places the computed Statistic data into the saving buffer. This is the Place Data thread. In order to preserve the Statistic structure consistency, this region of memory should not be accessed at the same time by two concurrent operations. It is a "critical region". Therefore, the two operations, the computation of Statistic and placing of Statistic data into the saving buffer are synchronized over this region by means of a mutex mechanism made from semaphores.

From the saving buffer the Statistic data is taken by the Database handling thread and is saved into the database. The saving buffer is a critical region using the implementation of the well-known Producer Consumer algorithm. Therefore Placing Data thread and Database Handling thread will never have access to the saving buffer at the same time. There is one Placing Data thread for each call but only one Database Handling thread, so the saving buffer has multiple producers and one consumer.

The diagram from appendix A.7 shows the action flow of these threads as well as the communication and synchronization between them using Petri-Nets.

The sending thread belonging to the Sender, contains a continuous loop in which the sending function is called at a fixed rate to send RTP packages to the Reflector. This one receives them, computes the parameters and updates the Statistic when it gets access to the critical region that is the Statistic data. When the Sender has finished sending a certain number of packets it resumes the Place Data thread. This thread places the Statistic data into the saving buffer when it gets access to the critical region. From the saving buffer the Database Handling thread saves it into the data-base.

3.3 Packets Transmission Mechanism

This section describes the mechanism implemented in the Sending thread. In VIP-Sim, like in all the other voice over IP applications the data packets need to be sent with a certain fixed rate. The mechanism implemented in the Sending thread is responsible to assure the constant sending rate of the data packets during the session of measurements. This mechanism keeps only the fixed rate, because the actual sending is performed by the *RTPUDPSession* object (see class diagram description 3.1) that also is responsible with the content of the packets and with its processing.

Therefore the mechanism consists of a loop that runs continuously during the session of measurements. Inside this loop the "Send" function from the *RTPUDPSession* object is called to send the packets. In order to keep a certain fixed interval between transmissions, a timing mechanism needs to be used.

There are several methods to implement the timing mechanism that offers the fixed time interval between transmissions:

- Making the thread to sleep for the desired time interval between transmissions.
- Using the operating system's timer (SetTimer). When this one times out it sends a message to the application or runs a callback function, that determines the sending of a packet.
- Reading the time continuously in the loop and decide locally whether it is time to send the packet or not [9].

As shown in the section 4.1, the sleep, SetTimer and timeSetEvent methods provided by the operating system can not provide the desired timer functionality due to two reasons. On one hand their resolution is not small enough and thus they cannot define all the time intervals desired. As an example think that with a 1 ms resolution there is no way to have a 32.5 ms time interval. On the other hand they are not accurate enough so they cannot provide the fixed rate with the desired accuracy. Therefore the third method is used to obtain the fixed sending rate. For the time reading, the most accurate and with the highest resolution time function is used, that is the Performance Counter. It has a resolution of less than 1 micro seconds and an accuracy around 1 microsecond.

The *RTPUDPSession* object contains a variable that keeps the value of the desired time interval between the transmissions of the packets or transmission rate (*time_interval*). A new variable is defined that keeps the time value of the next moment when a packet needs to be sent (*next_time*). After each sending this value is updated to point the next sending moment:

$next\_time = next\_time + time\_interval$

When the current time is read from the system (*current_time*, the value of the performance counter) it is compared with the next sending time (*next_time*). If it is equal (or greater) then the time interval has elapsed and a new packet is sent. After this the time of sending (*next_time*) is updated and the loop continues. In the figure 3.1 the loop diagram describes this algorithm.

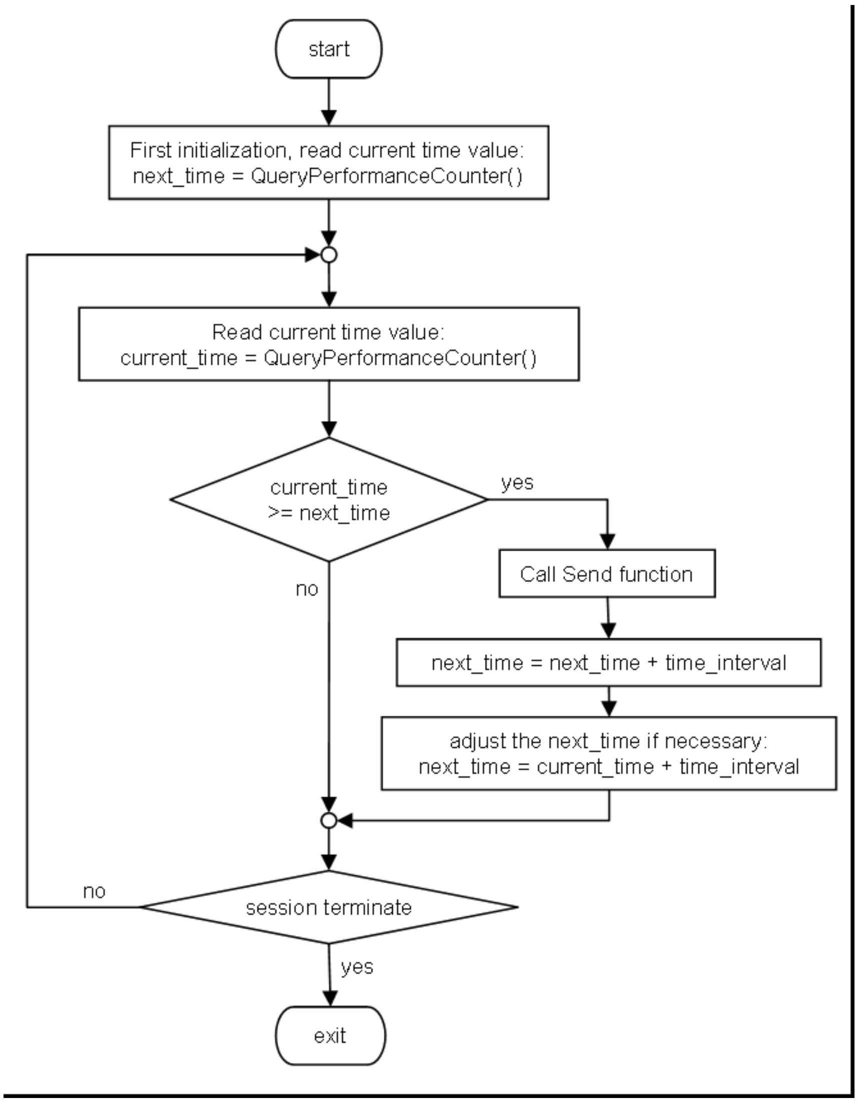

start
First initialization, read current time value:
next_time = QueryPerformanceCounter()
Read current time value:
current_time = QueryPerformanceCounter()
current_time >= next_time
yes
no
Call Send function
next_time = next_time + time_interval
adjust the next_time if necessary:
next_time = current_time + time_interval
no
session terminate
yes
exit

Figure 3.1: The main loop of the packets transmission mechanism.

When the comparison between the *current_time* and the *next_time* is performed the values may be equal meaning that the *current_time* is exactly the time to send the packet, or the *current_time* may be greater, meaning that the time to sent has been overpassed and the packet sending is late. The difference between *current_time* and the *next_time* gives in this case the **transmission timing deviation**, or the precision of the sending rate. This value is desired to be as small as possible, normally equal to 0. In VIPSim this difference is measured and is reported as the mean transmission timing deviation and maximum transmission timing deviation (see section 5.2.4 for results).

If the transmission timing deviation is greater than the *time_interval* it means that the sending of one packet has been missed. In other words, a frame has been skipped. This behaviour is not desired but it may happen due to too much stress on the operating system, the one who schedules the execution of all threads. When this occurs the adjustment of the next sending time is required accordingly to the following formula in order to have the next sending time in the future (greater than *current_time*) and not in the past (less than *current_time*):

next_time = current_time + time_interval

The number of skipped frames is also counted and reported by VIPSim (see section 5.2.4 for results).

The measured values of transmission timing deviation and skipped frames shows when sending problems occurs.

3.4 Multiple Calls

As mentioned in section 2.1 VIPSim is able to establish multiple calls. VIPSim may communicate with many Reflectors at the same time, and more than that, it can establish more than one call with each one of them. Even when the calls are established between one Sender and one Reflector they are treated as separate calls, having separate communication channels and separate bandwidth.

Handling multiple calls at the same time means that only the RTP packets belonging to all calls are sent and received concurrently. The call establishment procedures are performed sequential, separately for each call. This is done in order to be able to measure the establishment time for each one of them separately without one to interfere with another. Only after all calls have been established, the RTP packets start to be sent between Sender and Reflector and the measurements start to be performed.

Because of the fact that only RTP data transmission is performed in parallel, only the results regarding the network packets are influenced by the number of calls, and not the call establishment time.

Chapter 4

Efficiency Aspects

This section describes some aspects related to optimization of some operations performed in VIPSim, like timing, packets transmission mechanism, and parallel operations.

4.1 Timing in windows

As in any application that requires high precision and high resolution timing, in VIPsim there are two aspects related to timing that demand more consideration. One aspect regards the reading of the current time value, and the second is related to a timer mechanism. A timer is a mechanism that generates a signal (event or message) or calls a callback function when a certain time interval has elapsed.

In order to measure processes that may take very little time (< 1 ms) a very accurate time function that retrieves the time value with a very high resolution is needed. The operating system on which VIPSim runs offers several time functions [11]:

time() It is a standard OS routine that retrieves the time measured as the number of seconds elapsed since midnight (00:00:00), January 1, 1970. Include : WINBASE.H, Library : KERNEL32.LIB.

ftime() Gets the current time. It fills a structure that contains the time in seconds since midnight (00:00:00), January 1, 1970, coordinated universal time (UTC), and fraction of a second in milliseconds.

clock() Calculates the processor time used by the calling process. It returns the number of processor timer ticks that have elapsed. A timer tick is approximately equal to 1 ms.

GetTickCount()/GetCurrentTime() They are similar and each returns the number of milliseconds (+/- 55 milliseconds) since the user started Windows. Include : WINBASE.H, Library : KERNEL32.LIB.

timeGetTime() The timeGetTime() function retrieves the system time, in milliseconds. The system time is the time elapsed since Windows started. It belongs to Multimedia Timer Functions library. Include: MMSYSTEM.H, Library : WINMM.LIB.

QueryPerformanceCounter() The QueryPerformanceCounter function retrieves the current value of the high-resolution performance counter, if one exists. Most modern processors have such a Performance Counter that can provide a resolution less than 1 microsecond. This value counts the passed time since the system startup. Include: WINBASE.H, Library : KERNEL32.LIB.

The performance counter value increases with one unit several times per second. The function, QueryPerformanceFrequency returns a number showing how many times the performance counter increases per second. If the performance counter value is divided with the frequency value, the number of seconds since the system started is obtained. This is a real number having the precision of less then 1 microsecond. The integer part of it shows the number of seconds and the fraction part shows time less than 1 sec. The Performance counter value may not be used to retrieve the absolute current time, but is very good for measuring time intervals (even very short ones).

The functions presented above offer just the time reading services. The operating system also offers mechanisms for timer services.

sleep() The operating system offers a sleep function that suspends the execution of the current thread for a specified interval. If used in a loop it may create a periodical behaviour, or, in other words an operation that is performed periodically with a certain rate. The time interval is specified in milliseconds, so, its resolution may not be less than 1 millisecond.

SetTimer() The SetTimer function creates a timer with the specified time-out value. Again the time-out value is specified in milliseconds so the resolution is not less than 1 ms. When the timeout value elapses, it calls a callback function, or posts a message in the application message queue.

timeSetEvent() Belongs to the same library like timeGetTime(), which is the Multimedia Timer Functions library. The timeSetEvent function starts a specified timer event. When the timeout value elapses, it calls a specified callback function or sets or pulses a specified event object.

There are three values that characterize a time function.

- the resolution - the minimum time interval that the function may be requested to measure.
- the accuracy - shows how precise the measurement is, the maximum error between real (expected) and the measured value.
- response time - how much time the execution of the function takes.

Based on these values one may decide which time function is the best to be used in certain applications. If very short time intervals are measured, like in VIPSim's case, a time function with a very high resolution is needed. If the time function is required in a very tide loop then one with a short response time is needed. In all cases the accuracy is important. The more precise the better the function is.

The table 4.1 shows the characteristics of the time functions mentioned before, both time reading and timer services functions ([8] or see the testing algorithm in appendix A.9).

Function	**Resolution**	**Accuracy**	**Execution time**
Sleep	1 ms	10 ms	-
SetTimer	1 ms	10 ms	-
clock	1 ms	10 ms	-
ftime	1 ms	10 ms	-
getTickCount	1 ms	5-55 ms	0.1 us
timeGetTime / timeSetEvent	1 ms	1 ms	8.0 um
PerformanceCounter	$<$ 1 us	1 us	4.2 us

Table 4.1: Time functions characteristics (resolution, accuracy, execution time).

As may be seen from the table 4.1 all the standard OS functions have a resolution of 1 ms but they give the results with an error around 10-55 ms. The best resolution and the best accuracy may be obtained with the Performance Counter. Because in VIPSim time intervals that are measured may be very small (packets delays, interarrival time) the use of this function is the best solution. Using Performance Counter, the sending rate (time interval between two consecutive packets) may be defined with a resolution of microseconds, having 31.25 ms for example instead of 30,31 or 32 ms that may be obtained with other functions. On the other hand the precision is very high compared with the error of 10-55 ms that is given by the other functions. All these values may vary depending on the load of the operating system, which means how many processes it has to schedule.

The Performance Counter does not provide a timer mechanism but only a reading function. Therefore a timer needs to be implemented based on this function.

A timer is a service that gives notice to the application of the elapse of a time interval. If this service is not available it may be created using a simple time reading function in a continuous loop [9]. Inside this loop with a simple comparison that involves the current read time value and the value of the desired time interval, it can be determined if the time interval has elapsed. The desired action may be then performed. The algorithm is very simple and is presented in the following lines:

```
QueryPerformanceCounter(start_interval); //read the current time as the start of time interval
do //start loop
{
    QueryPerformanceCounter(current_time); //read the current time value
    elapsed_time = current_time - start_interval;
    if (elapsed_time >= time_interval)
    {
            QueryPerformanceCounter(start_interval); //start a new interval
            perform_the_required_action();
    }
} while (1);
```

This algorithm is used to determine the execution of the required action at a certain time interval that may be defined with a resolution of less than 1 microsecond and accuracy around the same value (1 us). As mentioned above the execution rate of the specified action may be delayed by the operating system when there are many processes that need to be scheduled for execution. That is why this algorithm gives best results on a system that is not very loaded. It helps if the thread running this loop has a real-time priority, but only in case there are not so many applications running in parallel with the application that implements this algorithm. VIPSim uses this algorithm for sending data packets (see section 3.3).

4.2 Distributed vs. Burst Transmission

In order to send the data packets on the network with a certain fixed rate the algorithm described in section 3.3 is used (see figure 3.1 on page 36). This section, describes how the time-intervals are measured and how after the elapse of each one of them the sending function is called to send the data packets. A graphical representation of the behaviour of this algorithm may be seen in figure 4.1. As seen, the time intervals of length "t" are shown starting from time 0 (0, t, 2t,...) as well as the moments when the packets are sent (black dot), which is the elapse of each time interval. This figure represents the sending sequence of packets for a one call measuring session. Only one packet is sent at the elapse of one time interval.

If there is a session with multiple calls, the easiest way to handle their packets transmissions is to change the sending of a packet when the time interval elapses, with a sequence of sendings of packets for all calls. In this case the behaviour of the sending mechanism will look like in the figure 4.2.

In figure 4.2 may be seen that when the time interval elapses, the packets, one for each call, are sent in a sequence one after the other, without break between transmissions. This creates an inconvenience because the actual sending of the packets on the network does not take place exactly at the same time when the application issues several Send commands in a sequence without breaks. They are buffered into

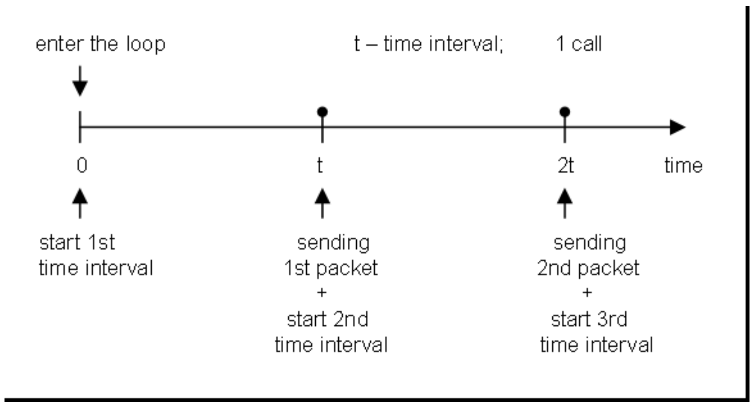

enter the loop
t – time interval;
1 call
0
t
2t
time
start 1st time interval
sending 1st packet + start 2nd time interval
sending 2nd packet + start 3rd time interval

Figure 4.1: Transmission of packets in one call

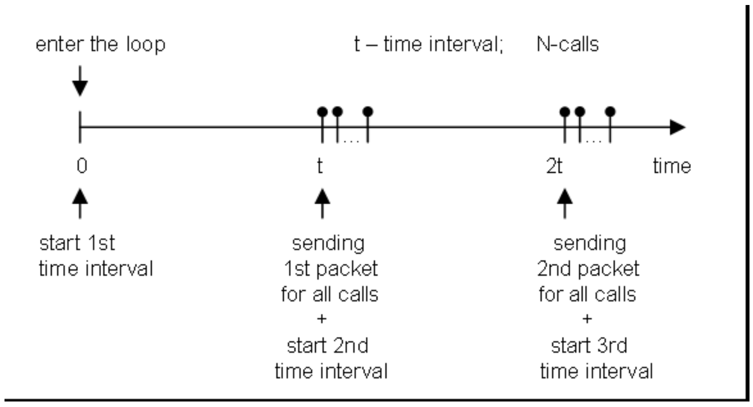

enter the loop
t – time interval;
N-calls
0
t
2t
time
start 1st time interval
sending 1st packet for all calls + start 2nd time interval
sending 2nd packet for all calls + start 3rd time interval

Figure 4.2: Transmission of packets for multiple calls - burst sending

the network buffer by the send command, from where they are sent by OS further on the network. The actual sending on the network is slower than the placing of the packets into the sending buffer. Therefore this burst sending process introduces some delay such that the last packet sent may have up to 10-20 ms extra delay than the first one (see section 5.2.1 for results of these tests).

In order to solve this problem, the packets sending process is distributed over one time interval. In this way a small break between transmissions of any single packet (not belonging to the same calls) is introduced. The rate between packets belonging to the same call is maintained and the buffering problems like before are eliminated, because the operating system has enough time between transmissions of a single packet to handle it properly.

The time interval is divided into smaller time intervals for handling single packets belonging to each call. This behaviour may be seen in the figure 4.3. In this figure there are 4 calls named A, B, C and D. Each time interval is then divided in

4 smaller time intervals, such that at the end of each one of them the sending of a packet belonging to one of the calls is handled. As may be seen from the picture, the rate between packets belonging to the same call is still as in the other cases. Now there is a small break between transmissions of single packets which offers the operating system enough time to perform the actual sending.

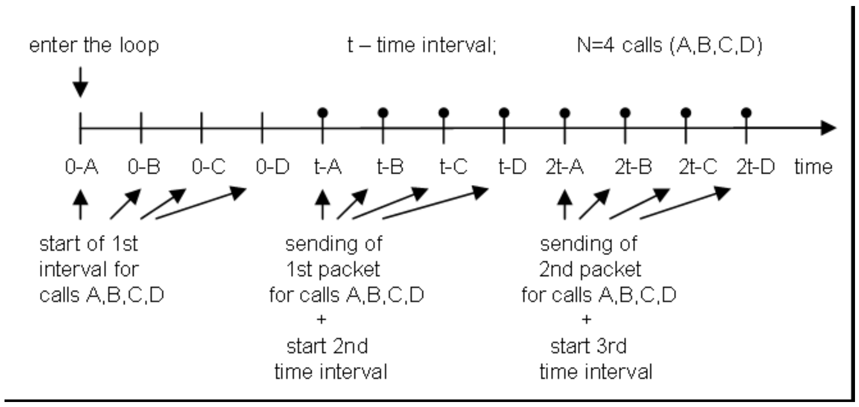

enter the loop
t – time interval;
N=4 calls (A,B,C,D)
0-A
0-B
0-C
0-D
t-A
t-B
t-C
t-D
2t-A
2t-B
2t-C
2t-D
time
start of 1st interval for calls A,B,C,D
sending of 1st packet for calls A,B,C,D + start 2nd time interval
sending of 2nd packet for calls A,B,C,D + start 3rd time interval

Figure 4.3: Transmission of packets for multiple calls - distributed sending

Using the last method, the operating system's sending function does not influence the delay between packets (round trip delay between Sender and Reflector). All the sent packets have almost the same delay (also depending on other factors), even if there is a multiple call session.

4.3 Parallel Operations

Upon receiving back a data packet the information about sequence number and sending time is extracted from its payload. It is used further in computation of the Statistics parameters. These Statistics are computed from the information contained in packets that arrive during a certain time period. After this time period elapses, the Statistic becomes inactive and later it is closed and becomes "ready to save".

The data from a "ready to save" Statistic needs to be saved into the database after every elapsed time period, for future use. The operations with the database: connection, data transmission and updating of the tables are time consuming operations that also depend on the network load. Therefore saving data may be time consuming. As seen in appendix A.8 and described in section 3.2, the number of packets sent, counted in the Sending thread, determines the start of saving process. Therefore the saving of data may influence the sending rate of the data packets.

This saving may also influence the reception of the packets by blocking the access to the critical region 1 (see A.8) more than it is required. Delaying the sending or the reception of the packets may influence the results by introducing more delay into the packets transmission (round trip delay). This problem occurs if the saving of data is performed in the same thread with the packets sending and if the access to the critical region takes longer than permitted (more than the interarrival time between two data packets).

The solution to this problem is to perform the saving of the Statistic data in a separate thread. As seen in the appendix A.8 there is a separate saving thread that saves Statistic data from a buffer into the data-base. The Statistic data is placed into this buffer by another thread called "Place Data" thread. There is a "Place Data" thread for each Statistic but the "saving" thread is only one for the whole session. This was done in order to avoid the concurrent access to the database when several Statistics (belonging to several calls) wanted to be saved at the same time. In this way they are only placed into a queue from where they are saved sequential by the saving thread.

The placing of Statistic data into the queue is not a copy operation of a memory zone from one place to another, but it is the placing of the address (pointer) of the Statistic into the queue . In this way the time needed to keep the critical region blocked is very small, so the influence of the reception of the packets is almost negligible.

As a conclusion, for better results, the saving operation is placed into a separate thread than the sending thread. On the other hand the saving operation is centralized not to have concurrent access to the database from all the Statistics. This centralization is realized by having only one saving thread and multiple "Place Data" threads that also keep the critical region blocked for a very short time period without influencing the rest of the application.

Chapter 5

Tests and Results

This section shows and describe results obtained from measurements performed with VIPSim. Some results are used to determine VIPSim's limitations and to find out the parameters' values that assure a proper functionality of VIPSim.

5.1 Testing plan

In order to observe VIPSim's behaviour and to determine if the results obtained from its measurements have real meaning some tests have to be made. Two sets of tests are performed. First set of tests regards the application characteristics, monitoring VIPSim's behaviour. In the second set of tests VIPSim is used to perform real measurements on a real environment.

1st Set of Tests

The first set of tests is intended to check the performances of VIPSim:

- the transmission precision.
- the delays introduced by the implementation.
- its limitations by determining the values allowed in order to have a normal operation
  - maximum number of calls
  - maximum packet size
  - minimum packets transmission rate

In order to test all theses factors, the environment VIPSim runs in should be ideal, and meaning that it should not influence the measured parameters. In an ideal environment all the delays due to OS, network and network devices should be 0. This is not possible to be obtained in normal conditions, so all these tests are performed in an "almost ideal" environment.

This "almost ideal" environment is created by reducing the impact of all the components of the environment. The OS has a big impact on the operation of any application through its multitasking scheduler. Therefore the number of applications including services and background application that run on the OS during the measurements is reduced very much. In this way the impact of the OS remains mainly due to network operations to serve VIPSim. These operations are packetization, buffering and network write/read.

To reduce the impact of the network, a very simple network is used without extra devices, like routers, firewalls and gatekeepers. The measuring system is made from two computers connected between them with a network cable. The round trip delay of the signal should be very small and it is considered 0 for the purpose of this test. Therefore, in these tests, the delays that appear are due to the application and the OS. Not having a dedicated OS, it is almost impossible to separate these two sources of delay. Therefore, VIPSim performances are strongly connected with the OS on which it runs. During these tests the measurements that involve a gatekeeper are not included.

The parameters that are changed during these measurements are:

- the number of calls, starting with one call and going up until the limit is reached
- the bandwidth to be filled by a call
- the packet size. Providing a certain bandwidth and the size of a packet, the application determines the rate at which to send packets for a certain call.

In order to measure the precision of the transmission (sending of packets) and the impact of the OS's sending function some extra parameters are introduced:

- mean and maximum transmission timing deviation - showing how much the real transmission moment is deviated from the expected transmission moment, the last one being computed as the last transmission timestamp plus the time interval between consecutive transmissions (rate). The mean value shows an average during the whole session and the maximum shows only the maximum value during this session.

  If T is the transmission rate and t1 is the last time when the transmission occurred then the next (expected) time when the transmission should occur is t2 = t1 + T. VIPSim continuously reads the current time value in a variable ct, and if this value is greater or equal to the next transmission time t2 (ct >= t2) then the transmission occurs again and the next transmission time (a new t2) is computed. The difference d = ct - t2 gives the transmission deviation. In an ideal case this value should be 0 so that the transmission occurs exactly at the expected moment.

- minimum, mean and maximum sending time - showing how much time it takes for the OS's sending function to execute. This execution time is measured by making the difference between the time value after the OS's send function returns and the time value before the OS's send is called. The reading of the two time values and the execution of the OS's send function are not atomic operations. The OS may interrupt the current process any time making the measured time to indicate more than the execution of the sending function. Therefore this value does not gives an exact information, so one should take into account that the real execution time is less or equal than the reported value.

2nd Set of Tests

The second set of tests are performed in order to measure the performances of the environment from a VoIP application point of view, which is the original purpose of VIPSim.

These tests consist of measurement sessions with one, up to N calls between two terminals as well as sessions with one or more calls between 1 Sender and M Reflectors. The network may contain routers, firewalls, and especially Gatekeepers. It is required that these tests should be performed on WAN not on local networks in order to measure a real environment that also involves Gatekeepers functionality.

If the Sender and Reflector are situated in separate local networks and they cannot see each other, the calls are made through gatekeepers, meaning that their traffic is routed through these gatekeepers. In this way the impact of the gatekeeper on the voice quality as well as on the time needed to establish the calls is measured.

Knowing the delays introduced by the application-OS entity from the 1st set of tests the impact of the network (including devices) over the quality of voice may be determined from the results obtained from this set of tests. If the impact of application-OS entity may be considered more or less constant in certain conditions, this 2nd set of tests gives different results from one network configuration (links plus devices) to another. The results show if a certain network configuration is good enough to carry on VoIP traffic. Neglecting the optimization of OS's performances that may be a problem of each user, these results may be used to enhance the performances of a network together with the network devices in order to provide better conditions for VoIP traffic.

5.2 Tests on an "almost ideal" network

5.2.1 Burst vs. Distributed Transmission test

The description of burst transmission and distributed transmission as well as the differences between them are presented in section 4.2 on page 42. In the current section (5.2.1) the numbers (measured data) that show the difference between the two types of transmissions are presented.

Two versions of the application are used. One having burst transmission implemented, meaning that theoretically the packets belonging to different calls are all sent at "the same time" and this repeated with a fixed rate. Practically they are not sent in parallel at the same time but in a tide sequence without delays between them like in the figure 4.2 on the page 44. The second version of the application has implemented a transmission mechanism that distributes the sending of packets belonging to separate calls over the time interval between successive transmissions of the same call, see figure 4.3 on page 44.

The results obtained with the burst transmission from 10 calls may be seen in table 5.1. This table contains two Statistics. Each Statistic contains the mean packets delay for the 10 calls. As may be seen, the first packet sent, belonging to the first call handled (having CRV = 55958), has the smallest delay. For the other calls the packets delay increases continuously so that the last call handled has a delay around 17 ms. This behaviour is due to the fact that the packets buffering is faster then the actual transmission on the network, so that the last packet buffered has to wait more in the queue until it is sent. Therefore the measured delay also includes the waiting time in the queue.

The next table, table 5.2, contains the delays from two Statistics with 10 calls handled with the distributed transmission algorithm. On this table may be seen that the mean packets delays are almost the same, around 2.1 ms, no matter what call it is (the first or the last). This is because the packets are not queued any more and because the OS now has enough time to handle each packet separately at the right time, due to the fact that their sending moment is distributed over a time interval.

As may be seen from these values as well as from description from section 4.2 the best solution is to use the implementation of the distributed transmission, in order not to introduce extra delays in packets transmission.

5.2.2 Number of calls and transmission rate limits

In this section results from the measurements that are used to determine VIPSim's limitations are presented. These limitations regards the number of calls that can be

CRV	Packets	Mean Delay
11492	20	17.65
30647	20	16.73
50170	20	15.67
3018	20	14.18
21666	20	13.71
43010	20	12.61
64141	20	11.14
18285	20	8.79
35642	20	9.08
55958	20	7.09
11492	20	17.06
30647	20	16.19
50170	20	15.88
3018	20	15.30
21666	20	14.62
43010	20	13.04
64141	20	11.83
18285	20	8.20
35642	20	8.54
55958	20	5.63

Table 5.1: Two Statistics of 10 calls showing the delays obtained using burst transmission algorithm.

establish, the maximum bandwidth and the packet size that can be used in order to have a normal operation, as well as the limitations imposed by different types of machines and OSs.

The two machines used are setup as follows: first the Sender is running on an Athlon-1GHz with Windows 2000 and the Reflector is running on an Athlon-500MHz with Windows 98 and then they are switched so that the Sender and the Reflector run on the oposite machine.

The table 5.3 shows the mean packets delays obtained from sessions having different number of calls and different transmission rates. The rows show the number of calls and the columns show bandwidths and respective the sending rates. The sending rate, expressed in ms, is determined from the bandwidth input, expressed in Kbytes/s, and a constant packet size used in all sessions (in the case of these measurements it is 1024 Bytes + headers). The transmission rate shown in table 5.3 is the distance between packets belonging to the same call. Because there are several calls and all packets belonging to them need to be handled there is an overall sending rate, involving all packets. The overall sending rate of all packets (not only

CRV	**Packets**	**Mean Delay**
56381	50	2.1654
4686	50	2.1654
20425	50	2.1921
35825	50	2.1678
49456	50	2.1715
65437	50	2.1691
6911	50	2.1301
16418	50	2.1477
29613	50	2.1617
43458	50	2.1352
56381	50	2.1407
4686	50	2.1384
20425	50	2.1447
35825	50	2.1479
49456	50	2.1382
65437	50	2.1600
6911	50	2.1793
16418	50	2.1614
29613	50	2.1835
43458	50	2.1615

Table 5.2: Two Statistics of 10 calls showing the delay obtained using distributed transmission algorithm.

one call) is lower, due to the usage of the distributed transmission algorithm (see section 4.2 on page 42). This overall rate may be seen in table 5.4. The overall transmission rate is obtained by dividing the transmission rate of packets belonging to the same call to the number of calls, distributing in this way the transmission for all packets over this interval. All data is expressed in ms. For example consider that for a call, the packets need to be sent every 200 ms. If there are 10 calls, a packet, one for each call, is sent every 200 / 10 = 20 ms.

	5 KB/s (200 ms)	**16 KB/s (62.5 ms)**	**20 KB/s (50 ms)**	**21 KB/s (47.6 ms)**	**24 KB/s (41.6 ms)**	**32 KB/s (31.25 ms)**
1	2.28	2.21	-	-	2.13	2.13
2	2.31	2.16	-	-	2.11	2.1
10	2.18	2.13	-	-	2.13	2.1
20	2.16	2.19	-	-	2.48	152.39
27	2.19	2.34	3.99	201.47	204.1	207.03

Table 5.3: Mean packet delays obtained on an Athlon 1GHz with Win 2K.

The next table, table 5.5, shows the mean delays obtained from sessions with dif-

	5 KB/s (200 ms)	**16 KB/s (62.5 ms)**	**20 KB/s (50 ms)**	**21 KB/s (47.6 ms)**	**24 KB/s (41.6 ms)**	**32 KB/s (31.25 ms)**
1	200	62.5	-	-	41.6	31.25
2	100	31.25	-	-	20.8	15.6
10	20	6.25	-	-	2.4	3.125
20	10	3.125	-	-	2.08	1.56
27	7.4	2.31	1.85	1.76	1.5	1.15
42	4.76	1.48	1.19	1.13	0.99	0.74

Table 5.4: Overall transmission rate computed by distributing the transmissions of all calls over the transmission interval of one call.

ferent number of calls and different transmission rates obtained on a system with Athlon-500MHz running Windows 98.

	5 KB/s (200 ms)	**16 KB/s (62.5 ms)**	**20 KB/s (50 ms)**	**21 KB/s (47.6 ms)**	**24 KB/s (41.6 ms)**	**32 KB/s (31.25 ms)**
1	12.14	12.9	-	-	12.54	12.6
10	4.82	12.88	-	-	12.09	12.14
20	6.77	12.09	-	-	12.49	1095.13
27	4.35	12.44	14.37	991.31	780.23	-
42	4.24	1320.03	-	-	-	-

Table 5.5: Mean packet delays obtained on an Athlon 500MHz with Win 98.

First observation is that when the Sender is running on the Athlon-1GHz/Win2k machine and the Reflector is running on the Athlon-500MHz/Win98 there is a limitation of 27 calls that can be established. When the machines on which the Sender and the Reflector are running are switched, there is a limit of 42 calls that can be established. This is because the limit of some OS's resourses is reached, which in this case are some network buffers (An error of "No buffer space is available" is received from the OS).

As may be seen from the table 5.3, in the most of the cases the mean packets delay is more or less constant, and has a value around 2.15 ms. For this machine (when the Sender runs on it), this value may be considered as the normal packets delay, representing the delay introduced by the implementation and the activity of the OS which in this case is Windows 2000. Whenever VIPSim operates in normal conditions the obtained delay is close to this value, depending of course on the input parameters (bandwidth, packet size).

It may also be seen in the table 5.3 that there is a limit from where the mean packets delay suddenly grows very much reaching 150-200 ms. Looking at the table 5.3 and the table 5.4 at the same time, one may see that whenever the distance between transmissions of two consecutive packets (no matter what calls they belong to) is less that a certain value the mean packets delay increases very much. This value is in this case around 1.80 ms. The application obviously does not run any more in the normal conditions. The conclusion is that this mean delay growth does not depend separately on the number of calls or on the bandwidth and packet size. It depends on the combination of the two factors. More precisely depends on the overall transmission rate of packets that should not be less than a certain value (1.8 ms in this case). The number of calls together with the bandwidth and packet size that determine a very small overall transmission rate, make VIPSim to reach its limit and to have an abnormally operation.

Reducing the packet size, for example to 512 bytes, one may observe that maximum overall transmission rate that can be reached in order to keep a normal operation is around 1.15 ms, smaller than 1.8 ms as in the case of a packet size of 1024 bytes. So, one may conclude that decreasing the packet size, also moves the border between normal and abnormally operation, by decreasing the permitted overall transmission rate.

Looking at the table 5.5, where the mean packets delays for the case when the Sender runs on the Athlon-500MHz/Win98 machine are shown, one may see that the mean delays are not more or less constant. They may vary from 4 ms up to 15 ms. These delays are obtained when VIPSim has a normal operation. Like in the previous case when the overall transmission rate drops under a certain value (1.8 ms), VIPSim starts to behave abnormally and the mean packets delay increases up to 1300 ms.

A conclusion that may be drawn from this, is that when the Sender runs on a Windows 98 OS and a slower machine (Athlon 500MHz) the mean delays are bigger, and that Windows 98 does not offer a constant behaviour (big variation of mean delay, 4 - 15 ms). The border between a normal operation and an abnormally operation of VIPSim is the same on the two machines, which is the overall transmission rate less than 1.8 ms, in case of a packet size of 1024 bytes.

The big differece of delays obtained on Windows 98, 4 to 17 ms may be due to OS's scheduling algorithm.

On one hand, one of VIPSim's limitations is the number of calls that it can establish due to the available OS's resources. On the other hand the combination of number of calls, bandwidth and packet size that determine a very small overall transmission rate is another limit.

Different tests made on different operating systems shows the impact of the operating system on the mean delay obtained. For example from another sets of tests made with a Sender running on a Windows ME operating system and a Reflector runing on a Windows XP operating system, a mean delay around 18-24 ms has been obtaned and a very big jitter (around 20). On the other hand, running a Sender on a Windows 2000 and a Reflector on a Windows XP operating system a much smaller delay, around 1.3, (even smaller than the 2.15 ms discussed above) has been obtained. All these were obtained from one call with a packet size of 1024 bytes and a bandwidth of 32 KB/s.

Also the maximum number of possible calls was bigger than 27 when using Windows 2000/Windows XP systems., and the minimum overall transmission rate is not around 1.8 ms as presented before in this section but much lower. An overall transmission rate of 1.66 ms has been reached without having an abnormal operation of VIPSim.

All these tests show the impact of the operating system VIPSim runs on. The operating system imposes many limits. The recommended operating system to run VIPSim on (Sender and Reflector) is Windows 2000 or Windows XP. There should be mentioned that any other Unix/Linux operating systems have not been tested.

5.2.3 Packet Size influence

This section shows the impact of the packet size on the mean packets delay.

The data from the table 5.6 shows the delays obtained from a sessions having one call and a bandwidth of 5 Kbytes/s (transmission rate = 200 ms) when using different packet sizes.

packet size (Bytes)	**delay (ms)**
64	0.51
128	0.63
256	0.9
512	1.44
1024	2.29
1025	36.99
1088	36.49
1280	36.22
1408	36.34
1536	37.29
1792	33.5
2048	33.88
2500	35.58

Table 5.6: Packet size influence over the mean delay.

As may be seen, the delay increases constantly with the increase of the packet size until a certain limit is reached. From this limit, which here is 1024 bytes the mean packets delay increases suddenly very much up to 33 - 38 ms. Usage of a big packet size introduce more delay. It is recommended to use small packet size in a voice transmission.

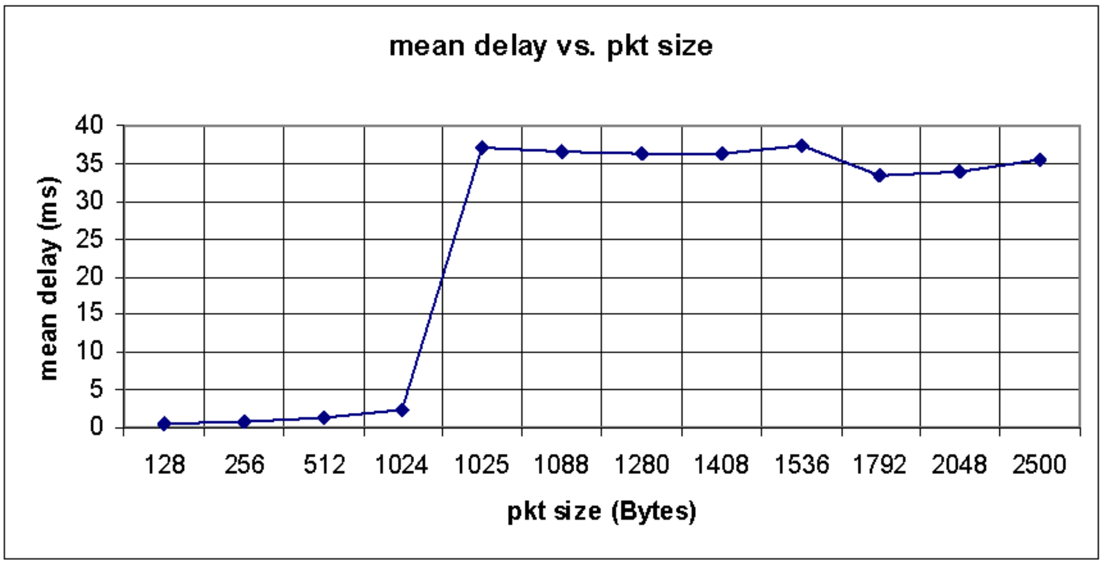


Figure 5.1: Mean Delay vs. Packet Size on Windows 2000. The packet size in the graph does not contain the UDP/IP headers.

5.2.4 Transmission Precision

One important issue is how precise the transmission mechanism is. This section shows some numbers related to this issue.

The table 5.7 shows the transmission timing deviation and the sending time obtained on an Athlon 1GHz/Win 2K system, and table 5.8 shows the transmission timing deviation and the sending time obtained on an Athlon 500MHz/Win 98 system.

As may be seen from these tables as well as from the charts from the appendix B.1 made based on the data from these tables, generally the mean transmission timing deviation is higher on the Windows 98 operating system ($<$ 0.08 ms) than on the Windows 2000 operating system ($<$ 0.01 ms). The maximum transmission timing deviations are very close (1.5 - 3.5 ms). Both the mean and maximum transmission deviations have small values, but less than the overall transmission rate so they do not influence the mean packets delay. Because the mean transmission timing deviation is very small (0.01 - 0.08 ms) in comparison with the mean packets delay and

	5 KB/s (200 ms)	**16 KB/s (62.5 ms)**	**24 KB/s (41.6 ms)**	**32 KB/s (31.25 ms)**
mean dev	0.0024	0.0053	0.0031	0.0044
max dev	0.24	3.0982	1.4114	2.7909
min snd tm	0.0285	0.0179	0.0173	0.0168
mean snd tm	0.1004	0.0641	0.041	0.0377
max snd tm	0.1774	1.1761	0.4339	1.1175

Table 5.7: Transmission timing deviation and transmission duration on an Athlon 1GHz with Win 2K.

	5 KB/s (200 ms)	**16 KB/s (62.5 ms)**	**24 KB/s (41.6 ms)**	**32 KB/s (31.25 ms)**
mean dev	0.04	0.0376	0.00609	0.00777
max dev	1.7784	3.2141	3.3155	3.3683
min snd tm	0.1039	0.0388	0.0939	0.0888
mean snd tm	0.1245	0.0473	0.1109	0.1103
max snd tm	1.8463	3.2141	2.9702	4.3061

Table 5.8: Transmission timing deviation and transmission duration on an Athlon 500MHz with Win 98.

the transmission rate, it may be considered that the transmission mechanism is precise enough for its functionality. Of course that the transmission timing deviation may increase on certain moments depending on the stress of the operating system, and its process scheduling algorithm.

Looking at the sending function (OS's sending function) execution time one may see that it takes more time on Athlon 500MHz/Windows 98 machine than on the Athlon 1GHz/Windows 2000 machine. In both cases the sending function execution time is very small compared with the transmission rate so it does not influence the transmission timing and the mean packets delay. Of course like in the previous case the OS's activity may influence this value too.

5.2.5 OS stress test

All the tests made to find out VIPSim's limits from section 5.2.2 were made in a non stressed operating system conditions. An operating system becomes stressed when several applications are running in parallel requiring much processor time and many resources. In this case the operating system responds very slow to applications' requests.

This section shows the difference between VIPsim's operation in a stressed and non stressed operating system. The next two tables, 5.9 and 5.10 show the packets lost (L), packets out of order (OO), jitter (J), packets delay (D) and interarrival time (I) from a session with one call and a bandwidth of 32 KB/s and packets size of 1024 bytes in both cases of non stressed OS (table 5.9) and stressed OS (table 5.10).

Pkt	L	OO	J	min D	mean D	max D	min I	mean I	max I
320	0	0	0.0000	2.0603	2.1565	2.6353	29.9935	31.2499	32.4424
320	0	0	0.0379	2.0634	2.1654	3.5599	28.4413	31.2495	33.9317
320	0	0	0.0000	2.0662	2.1564	2.7610	29.5613	31.2496	32.8193
320	0	0	0.0460	2.0670	2.1701	3.9687	29.4300	31.2493	33.1461
320	0	0	0.0000	2.0620	2.1574	2.6677	30.0994	31.2496	32.4944
320	0	0	0.0000	2.0617	2.1572	2.7755	29.0115	31.2498	33.3922
320	0	0	0.0019	2.0620	2.1787	6.7855	26.5372	31.2502	35.9155
320	0	0	0.0000	2.0642	2.1602	2.7881	30.6335	31.2501	31.8608
320	0	0	0.0000	2.0659	2.1623	3.2317	30.0667	31.2499	32.3047

Table 5.9: VIPSim operation on a non stressed OS. The table contains measured parameters related to the network packets.

Pkt	L	OO	J	min D	mean D	max D	min I	mean I	max I
320	0	0	0.0553	2.7138	3.0316	10.2985	21.0887	31.2494	41.4134
320	0	0	0.1803	2.7696	2.9976	5.3253	22.1729	31.2534	40.4711
320	0	0	0.1037	2.6855	2.9500	10.3879	13.7492	31.2493	49.7030
320	0	0	0.0026	2.6749	2.9002	7.5342	7.5149	31.2492	50.5659
320	0	0	0.0489	2.6936	3.0807	14.1007	6.5782	31.1972	55.8655
320	0	0	1.0238	2.6858	3.0612	10.3809	8.1421	31.2500	54.9827
320	0	0	0.4159	2.7677	3.1120	17.1290	0.8085	31.4506	89.2499
320	0	0	0.2470	2.3003	3.1095	16.1973	12.2071	31.8287	89.7561
320	0	0	0.1256	2.7070	3.0032	6.2857	8.4539	31.8552	84.1411
320	0	0	0.4814	2.7154	3.1014	17.9573	3.8580	31.9585	104.1090
320	0	0	0.0595	2.6819	2.9145	5.2674	14.6527	31.2497	47.9027
320	0	0	0.0524	2.2944	3.1439	42.9800	0.8082	31.6168	94.7741
320	0	0	0.2165	2.6582	2.8818	6.4469	20.9907	31.2497	41.4743
320	0	0	0.0024	2.6506	2.8729	5.4937	25.3236	32.0993	253.5467
320	0	0	1.2834	2.6833	3.0031	17.0902	12.1353	31.2487	58.0873
319	0	0	0.0329	2.6473	2.8046	5.4943	19.9947	31.2495	42.6915

Table 5.10: VIPSim operation on a stressed OS. The table contains measured parameters related to the network packets.

As may be seen, in the case of a non stressed operating system (table 5.9) VIPSim runs in normal conditions and the delay is more or less constant (around 2.15 ms) al the time. Also the jitter is very small almost equal to 0.0 and the interarrival time is generally 31.25 ms as resulted also from the usage of a bandwidth of 32 KB/s and packet size of 1024B.

In the second table 5.10, where the results from a stressed system are shown there may be seen a small increase of the mean delay, now being around 3 ms, and an increase of the jitter, in several cases going above 1. When the system is stressed the delay and the interarrival time are not constant any more and they vary from one Statistic to another, also determining the increase of the jitter as well. The delay may in some cases increase up to 42 ms and the interarrival time up to 250 ms as may be seen from the maximum values of delay and interarrival time from the table 5.10.

In a stressed operating system even the transmission timing deviation increases, so that the mean value is greater than 1 ms and the maximum value may be around 250 ms. The duration of the OS's sending function may take, in the case of a stressed system, more time (up to 40 ms). The increase of the transmission timing deviation shows that the transmission rate is not constant any more and that the packets have a delay even from the sending. Therefore the voice quality is lowered from the source. The long duration of the OS's sending function in some cases shows that a stressed system cannot serve the applications in the proper way in order to ensure a normal operation.

From this test there may be drawn the conclusion that a stressed operating system has a big negative influence over the packets transmission.

5.3 Tests on "real" network

These measurements show an example of VIPSim's usage. In these measurements the tested or monitored part is not VIPSim like in the 1st set of tests, but the environment that is measured. The target of these measurements is to find the characteristics of this environment.

This section does not present special cases and problems that can occur on real environments. Only as an example of VIPSim's usage the table from appendix B.2 is attached. The environment was a network with multiple devices and some extra traffic (DTU's network). The operating systems, on which VIPSim ran, were Windows 2000/Windows XP.

In this table a session with 30 calls, each with a 20 Kbytes/s bandwidth and a packet size of 1024 bytes is shown. There may be seen that even that the mean delay is more or less constant (around 1.45 ms) the maximum delays varies a lot (2-12 ms), indicating some extra traffic activity on the network. Therefore the jitter is also higher compared with the one obtained from tests on an "almost ideal" network.

The mean transmission timing deviation and the sending function execution time have very small values (less than 0.1 ms) indicating a good transmission precision.

Having a small jitter, a constant interarrival rate and a very small delay ($1.45 < 150$ ms) this system is suitable to run VoIP applications.

Chapter 6

Conclusions

This section offers the final conclusions of the study of a voice over IP quality of service measurement tool.

Knowing the QoS that can be offered by a system is important for consultants and engineers. On one hand, they can determine the behaviour of a future installation of a VoIP application on that system and on the other hand they can troubleshoot VoIP system already installed on different systems. VIPSim is a tool that can help them to do this job. It offers important information about parameters that characterize system's QoS such that VIPsim's users start to know and understand that system from a VoIP application's point of view.

VIPSim does not only provide information about the state of the QoS, but it can also be used to locate in time and to identify the bottlenecks problems. In this way one may know where to bring improvements to the system for better results.

The actual version of VIPSim is a measuring tool that reports the measured values. A user is needed to read and interpret these results. As future versions, VIPSim, or its base principle, can be implemented directly in VoIP applications. In this way a VoIP application will be able to monitor the QoS and adjust itself in order to provide better services. On the other hand, a similar tool may be implemented as part of the operating system, informing the user about the state of the QoS and maybe also triggering some actions meant to improve the quality.

Measuring VoIP quality is not the only way VIPSim can be used. It measures the operating system's impact and the network transmissions quality. Therefore the results obtained with VIPSim can be used to determine the behaviour of any other application that uses network tranmissions and needs a real-time response.

As this version of VIPSim is built now, the posibility of running a VoIP application at the same time with VIPSim on the same machine to observe the QoS state while experiencing the voice quality is not available. This is because this version of VIPSim uses too much the operating system's resources. Having a Real-Time priority for its threads, it uses almost 100% of the CPU, slowing down in this way the other applications that might run at the same time.

This thesis describes the first version of VIPSim. This version is concentrated more on offering the posibility of multiple calls between two endpoints (Sender and Reflector) than on optimizing the implementation from the usage of OS's resources point of view. Trying to optimize the implementation to use fewer resources (threads, sockets, buffers) is for a future development of VIPSim.

Chapter 7

Glossary

VIPSim - Voice over IP Simulations - application that measures the QoS that can be offered by a system from a VoIP point of view.

VoIP Voice over IP

QoS - Quality of Service

Delay (Latency) - Also called Round Trip Delay, is the time needed for the data packet to travel through the system to the destination and back.

Jitter - An estimate of the statistical variance of the RTP data packet interarrival time.

Packets Lost - Packets lost on the network.

Packets out of Order - Packets that arrive at the destination but not in the order they were sent.

Interarrival time - The time between consecutive packets that arrived at the destination.

Intersending time - The time between two consecutive sent packets.

Statistic - a data structure that containes measured/computed values like (delay,jitter,interarrival time, e.t.c.) for a certain time period.

Call Procedure measurements - Measurements related to call establishment and call ending. They containes counters for calls established correctly and failed, calls ended normal and abnormal, and call establishment time.

Voice Quality measurements - Measurements related to data packets. They contains information about packets' delays, jitter, interaarival time, packets lost and out of order.

Sender - VIPSim's part that sends packets to the Reflector and receives them back to perform the measurements.

Reflector - VIPSim's part that receives packets from the Sender and sends them back to it.

Bottleneck - A bottleneck is a congestion in a flow. Here is the congestion of bits flowing on the network.

CRV - Call Reference Value. A value intended to uniquely identify a call.

Producer/Consumer buffer (queue) - A buffer, or queue that does not allow concurrent access of several processes over its data. Producers provide input data and Consumers retrieve it as the output from this buffer.

Transmission timing deviation - Shows the deviation of the sending (transmission) moment of a packet from the expected (required) value.

Sending execution time - Is the time of execution of the operating system's sending function.

Appendix A

Graphs, Diagrams and Procedures

A.1 Call Establishment Procedures using H.225.0 and H.245 Messages

H.225.0 messages.

Setup message is sent between H.323 terminals in order to signal the desire of establishing a call.

Call Proceeding message is sent as an answer to the Setup message signaling that the called endpoint is processing the request.

Alerting message is sent by the called endpoint in order to signal that the phone is ringing.

Connect message is sent by the called endpoint in order to accept the call and to connect to signal that they are connected.

Release Complete message sent by any endpoint to notify the other party that it ended or wants to end the call. It may be also sent after Alerting to signal that it does not want to establish the call.

H.225.0 - RAS messages.

Gatekeeper Request - GRQ - message sent by a terminal to a gatekeeper when it wants to discover the existence of a gatekeeper.

Gatekeeper Confirm / Reject - GCF/GRJ - two messages sent by the gatekeeper as a reply to GRQ. The gatekeeper sends only one of them in order to accept and respectively to reject the GRQ.

Registration Request - RRQ - message sent by a terminal to a gatekeeper when it wants to register to a gatekeeper.

Registration Confirm / Reject - RCF/RRJ - two messages sent by the gatekeeper as a reply to RRQ. The gatekeeper sends only one of them in order to accept and respectively to reject the RRQ.

Admission Request - ARQ - message sent by a terminal to a gatekeeper when it wants to be allowed by the gatekeeper to access the packet-based network.

Admission Confirm / Reject - RCF/RRJ - two messages sent by the gatekeeper as a reply to ARQ. The gatekeeper sends only one of them in order to accept and respectively to reject the ARQ.

Disengage Request - DRQ - message sent by a terminal to a gatekeeper or by a gatekeeper to a terminal when it wants to cancel the admission to packet-based network.

Disengage Confirm / Reject - DCF/DRJ - two messages sent as a reply to DRQ. There is sent only one of them in order to accept and respectively to reject the DRQ.

Unregistration Request - URQ - message sent by a terminal to a gatekeeper when it wants to unregister from the gatekeeper.

Unregistration Confirm / Reject - UCF/URJ - two messages sent by the gatekeeper as a reply to URQ. The gatekeeper sends only one of them in order to accept and respectively to reject the URQ.

Location Request - LRQ - message sent by a terminal to a gatekeeper when it wants to translate some alias address or phone number of another terminal to an IP address.

Location Confirm / Reject - LCF/LRJ - two messages sent by the gatekeeper as a reply to LRQ. The gatekeeper sends only one of them in order to accept and respectively to reject the LRQ.

Bandwidth Request - BRQ - message sent by a terminal to a gatekeeper or by the gatekeeper to a terminal when it wants to change the bandwidth used in a call.

Bandwidth Confirm / Reject - BCF/BRJ - two messages sent as a reply to BRQ. There is sent only one of them in order to accept and respectively to reject the BRQ.

H.245 messages.

Terminal Capability Set - contains the video/audio or data capability of an endpoint. This message is used to inform the other endpoint about these capabilities.

Master Slave Determination - message used in the master/slave determination procedure. One endpoint needs to be a master in order to solve conflicts between endpoint involved in a connection.

Open Logical Channel - message signalling the desire of opening of a logical channel. It contains the data type of the information that is transmitted on this channel. The acknowledge message contains the address (IP and port number) where to send this information.

End Session - message signalling the termination of an H.245 session.

A.2 Master Slave Determination Procedure using H.245 Messages

This appendix describes the H.245 procedure of determining the master and slave between two terminals involved in a call. The following text is extracted from appendix C.2 of Recommendation H.245 - Control Protocol for Multimedia Communication.

C.2.1.1 Protocol overview - initiation by local user

A master slave determination procedure is initiated when the DETERMINE.request primitive is issued by the MSDSE user. A MasterSlaveDetermination message is sent to the peer MSDSE, and timer T106 is started. If a MasterSlaveDeterminationAck message is received in response to the MasterSlaveDetermination message then timer T106 is stopped and the user is informed with the DETERMINE.confirm primitive that the master slave determination procedure was successful and a MasterSlaveDeterminationAck message is sent to the peer MSDSE. If however a MasterSlaveDeterminationReject message is received in response to the MasterSlaveDetermination message, then a new status determination number is generated, timer T106 is restarted, and another MasterSlaveDetermination message is sent. If after sending a MasterSlaveDetermination message N100 times, a MasterSlaveDeterminationAck still has not been received, then timer T106 is stopped and the user is informed with the REJECT.indication primitive that the master slave determination procedure has failed to produce a result.

If timer T106 expires then the MSDSE user is informed with the REJECT.indication primitive and a MasterSlaveDeterminationRelease message is sent to the peer MSDSE.

C.2.1.2 Protocol overview - initiation by remote user

When a MasterSlaveDetermination message is received at the MSDSE, a status determination procedure is initiated. If the status determination procedure returns a determinate result, then the user is informed of the master slave determination result with the DETERMINE.indication primitive, a MasterSlaveDeterminationAck message is sent to the peer MSDSE, and timer T106 is started. If a MasterSlaveDeterminationAck message is received in response to the MasterSlaveDeterminationAck message, then timer T106 is stopped and the user is informed with the DETERMINE.confirm primitive that the master slave determination procedure was successful.

If timer T106 expires then the MSDSE user is informed with the REJECT.indication primitive.

If however the status determination procedure returns an indeterminate result, then the MasterSlaveDeterminationReject message is sent to the peer MSDSE.

C.2.1.3 Protocol overview - simultaneous initiation

When a MasterSlaveDetermination message is received at the MSDSE that itself has already initiated a status determination procedure, and is awaiting a Master-SlaveDeterminationAck or MasterSlaveDeterminationReject message, then a status determination procedure is initiated. If the status determination procedure returns a determinate result, the MSDSE responds as if the procedure was initiated by the remote user, and the procedures described above for this condition apply.

If however the status determination procedure returns an indeterminate result, then a new status determination number is generated, and the MSDSE responds as if the procedure was again initiated by the local MSDSE user as described above.

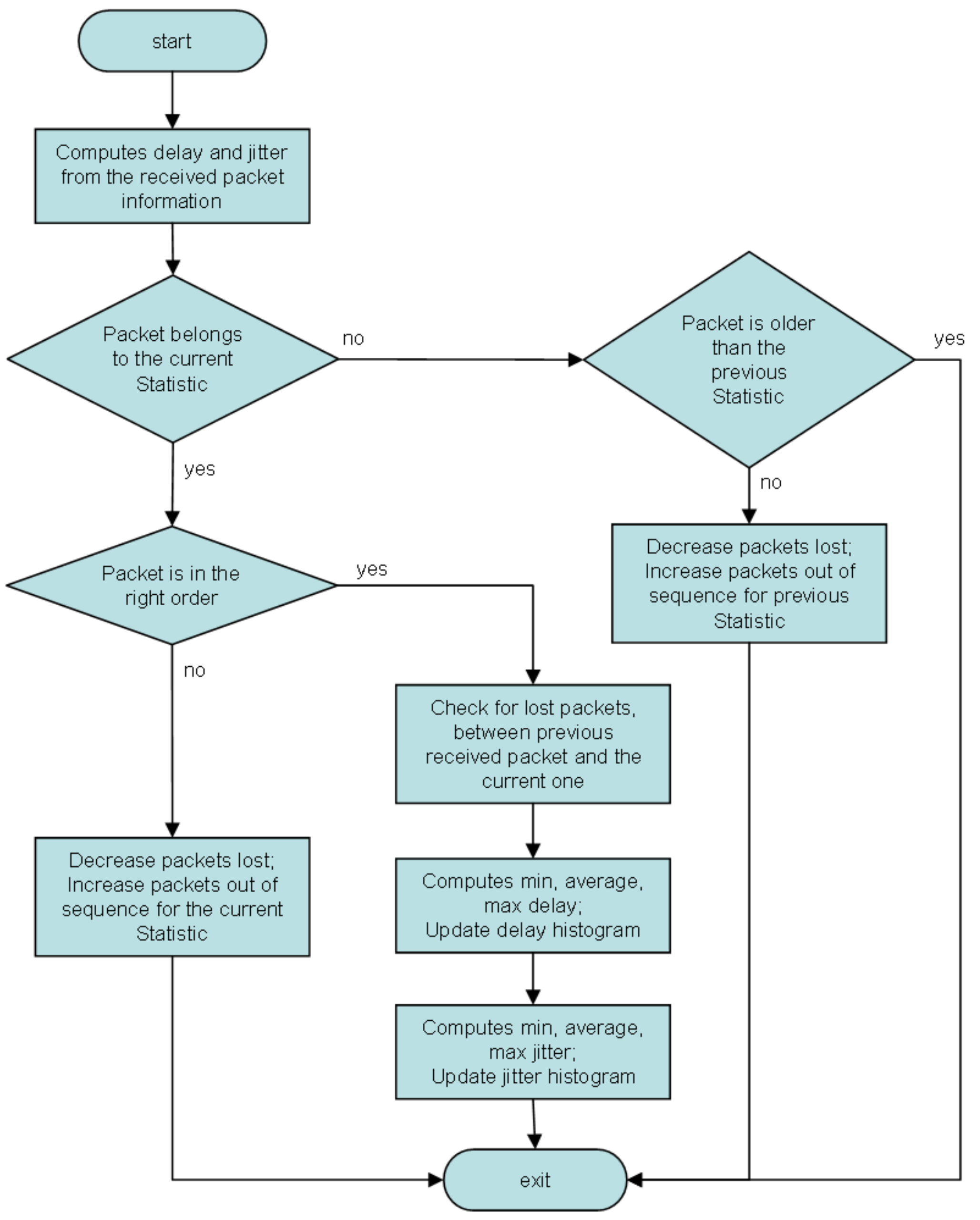
C.2.1.4 Status determination procedure

The following procedure is used to determine which terminal is the master from the terminalType and statusDeterminationNumber values. Firstly, the terminal-Type values are compared and the terminal with the larger terminal type number is determined as the master. If the terminal type numbers are the same, the status-DeterminationNumbers are compared using modulo arithmetic to determine which is master.

If both terminals have equal terminalType field values and the difference between statusDeterminationNumber field values modulo 2^{24} is 0 or 2^{23}, an indeterminate result is obtained.

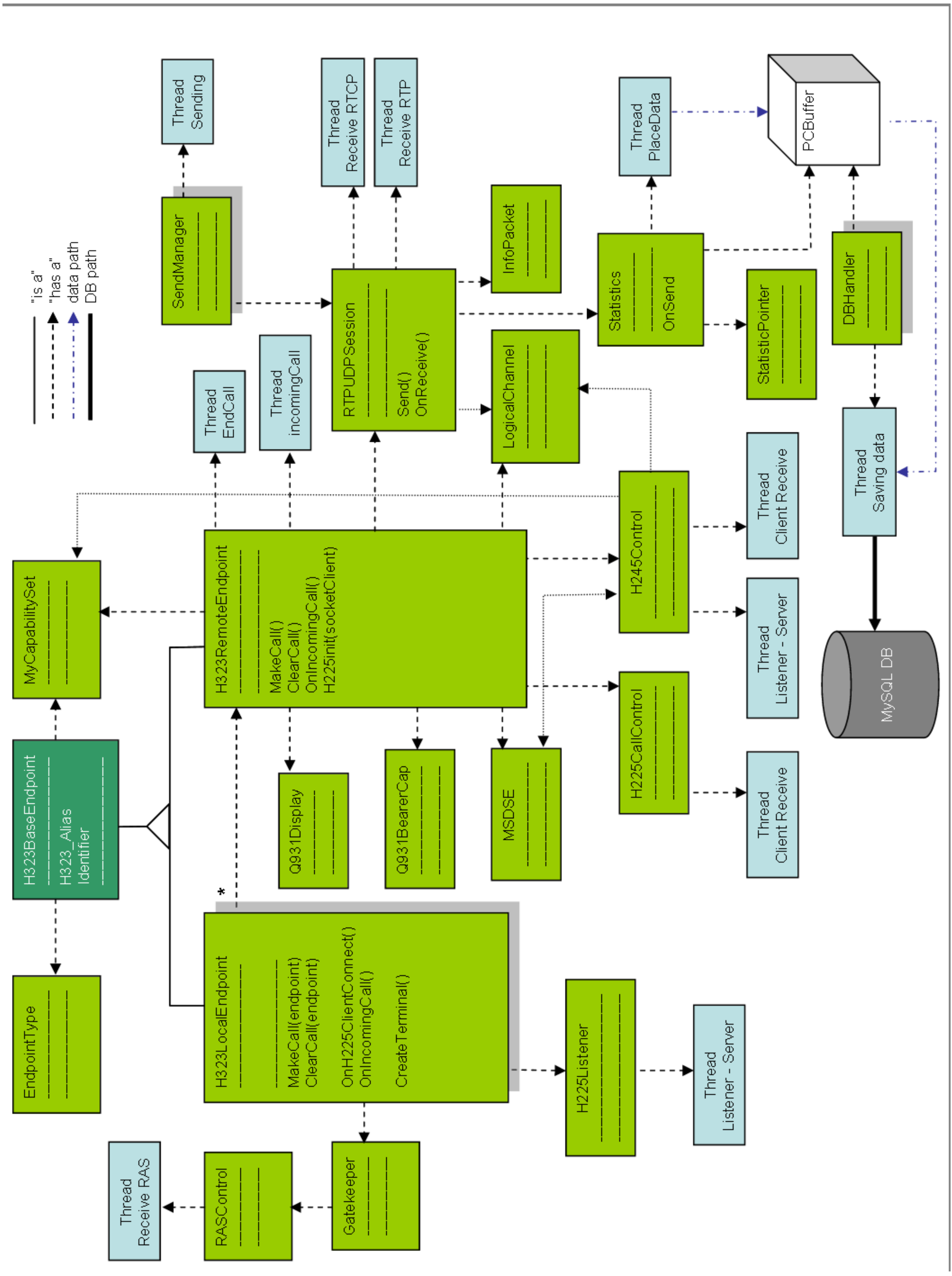
A.3 Measuring Algorithm Flowchart

This appendix shows the algorithm used to compute the Statistic data.


start
Computes delay and jitter from the received packet information
Packet belongs to the current Statistic
no
yes
Packet is older than the previous Statistic
yes
no
Decrease packets lost; Increase packets out of sequence for previous Statistic
Packet is in the right order
yes
no
Check for lost packets, between previous received packet and the current one
Computes min, average, max delay; Update delay histogram
Computes min, average, max jitter; Update jitter histogram
Decrease packets lost; Increase packets out of sequence for the current Statistic
exit

A.4 Class Diagram

This appendix contains VIPSim's class diagram.


"is a"
"has a"
data path
DB path
Thread Sending
SendManager
Thread Receive RTCP
Thread Receive RTP
RTPUDPSession
Send()
OnReceive()
InfoPacket
Thread EndCall
Thread incomingCall
LogicalChannel
Statistics
OnSend
Thread PlaceData
PCBuffer
StatisticPointer
DBHandler
Thread Saving data
MySQL DB
MyCapabilitySet
H323RemoteEndpoint
MakeCall()
ClearCall()
OnIncomingCall()
H225init(socketClient)
H245Control
Thread Client Receive
Thread Listener - Server
H225CallControl
Thread Client Receive
H323BaseEndpoint
H323_Alias Identifier
Q931Display
Q931BearerCap
MSDSE
EndpointType
H323LocalEndpoint
MakeCall(endpoint)
ClearCall(endpoint)
OnH225ClientConnect()
OnIncomingCall()
CreateTerminal()
*
H225Listener
Thread Listener - Server
Gatekeeper
RASControl
Thread Receive RAS

A.5 Jitter Calculation

The following definition is extracted from Recommendation H.225.0 - Call Signaling Protocols and Media Stream Packetization for Packet-Based Multimedia Communication Systems - Appendix A.6.3.1

interarrival jitter: 32 bits. An estimate of the statistical variance of the RTP data packet interarrival time, measured in timestamp units and expressed as an unsigned integer. The interarrival jitter J is defined to be the mean deviation (smoothed absolute value) of the difference D in packet spacing at the receiver compared to the sender for a pair of packets. As shown in the equation below, this is equivalent to the difference in the "relative transit time" for the two packets; the relative transit time is the difference between a packet's RTP timestamp and the receiver's clock at the time of arrival, measured in the same units.

D(i,j) = (Rj-Ri)-(Sj-Si) = (Rj-Sj) - (Ri-Si)

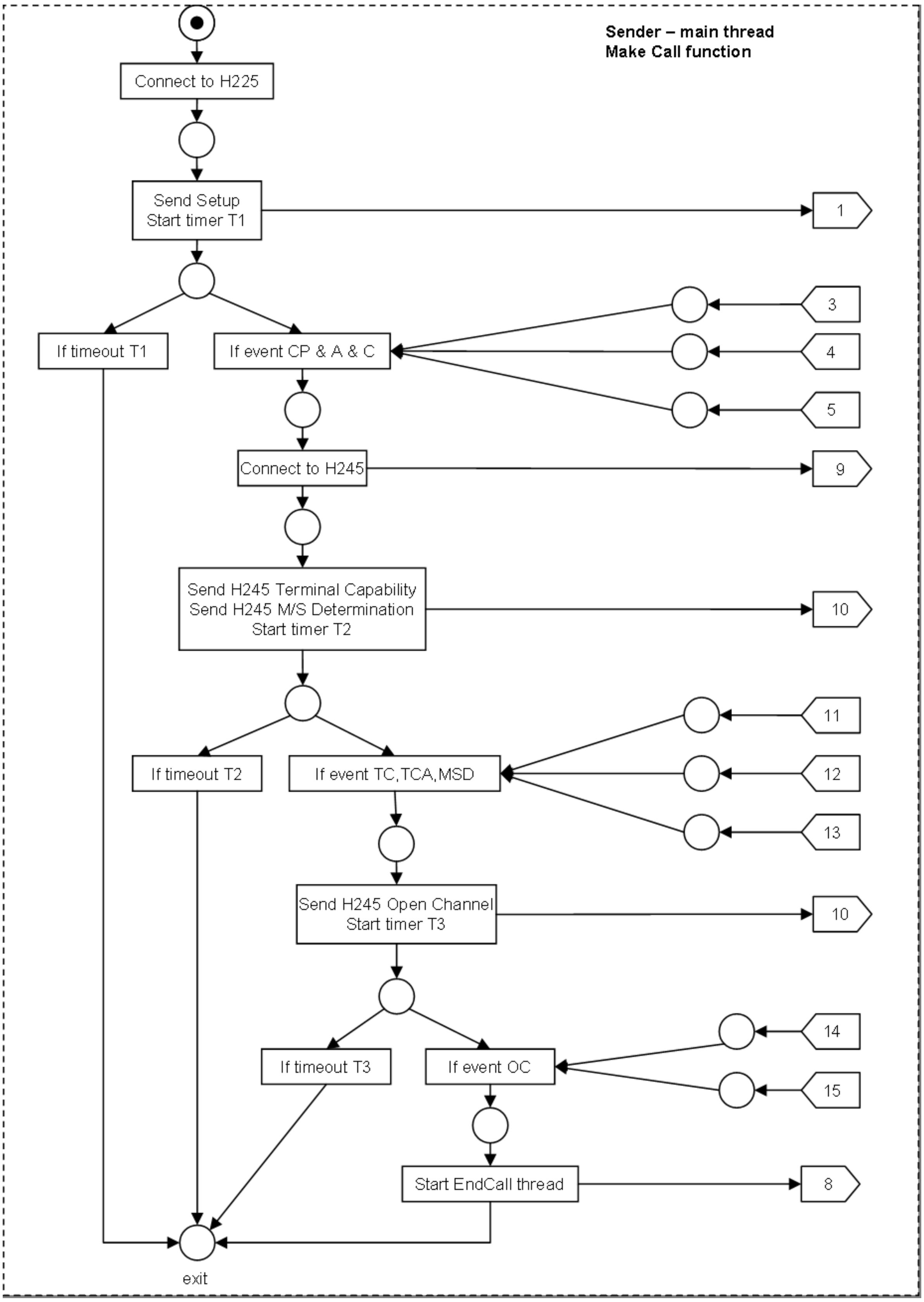
The interarrival jitter is calculated continuously as each data packet i is received from source SSRC_n, using this difference D for that packet and the previous packet i - 1 in order of arrival (not necessarily in sequence), according to the formula:

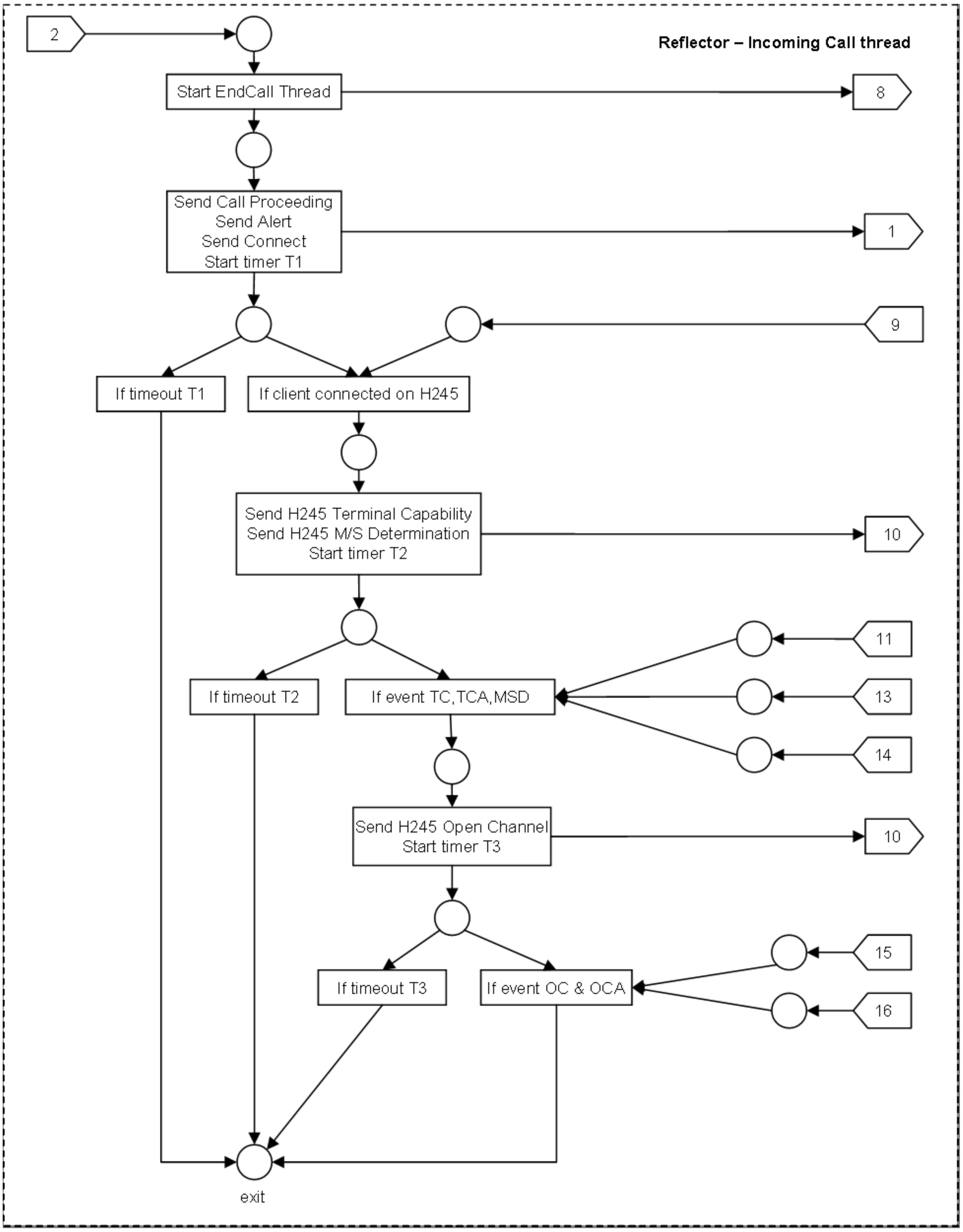
$J = J + \frac{|D(i-1,i)|-J}{16}$

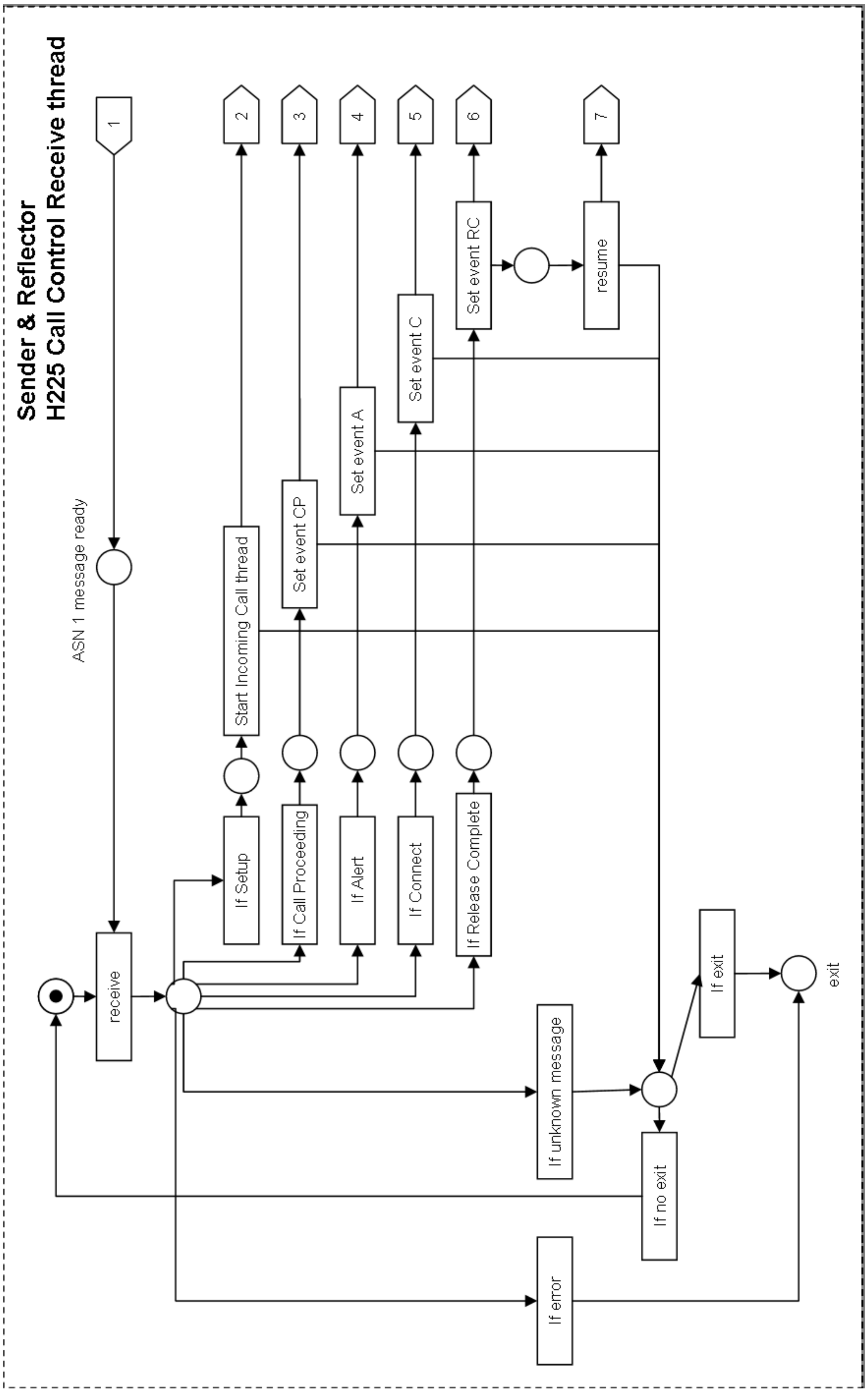
Whenever a reception report is issued, the current value of J is sampled. The jitter calculation is prescribed here to allow profile-independent monitors to make valid interpretations of reports coming from different implementations. This algorithm is the optimal first-order estimator and the gain parameter 1/16 gives a good noise reduction ratio while maintaining a reasonable rate of convergence [A-9]. A sample implementation is shown in A.8.

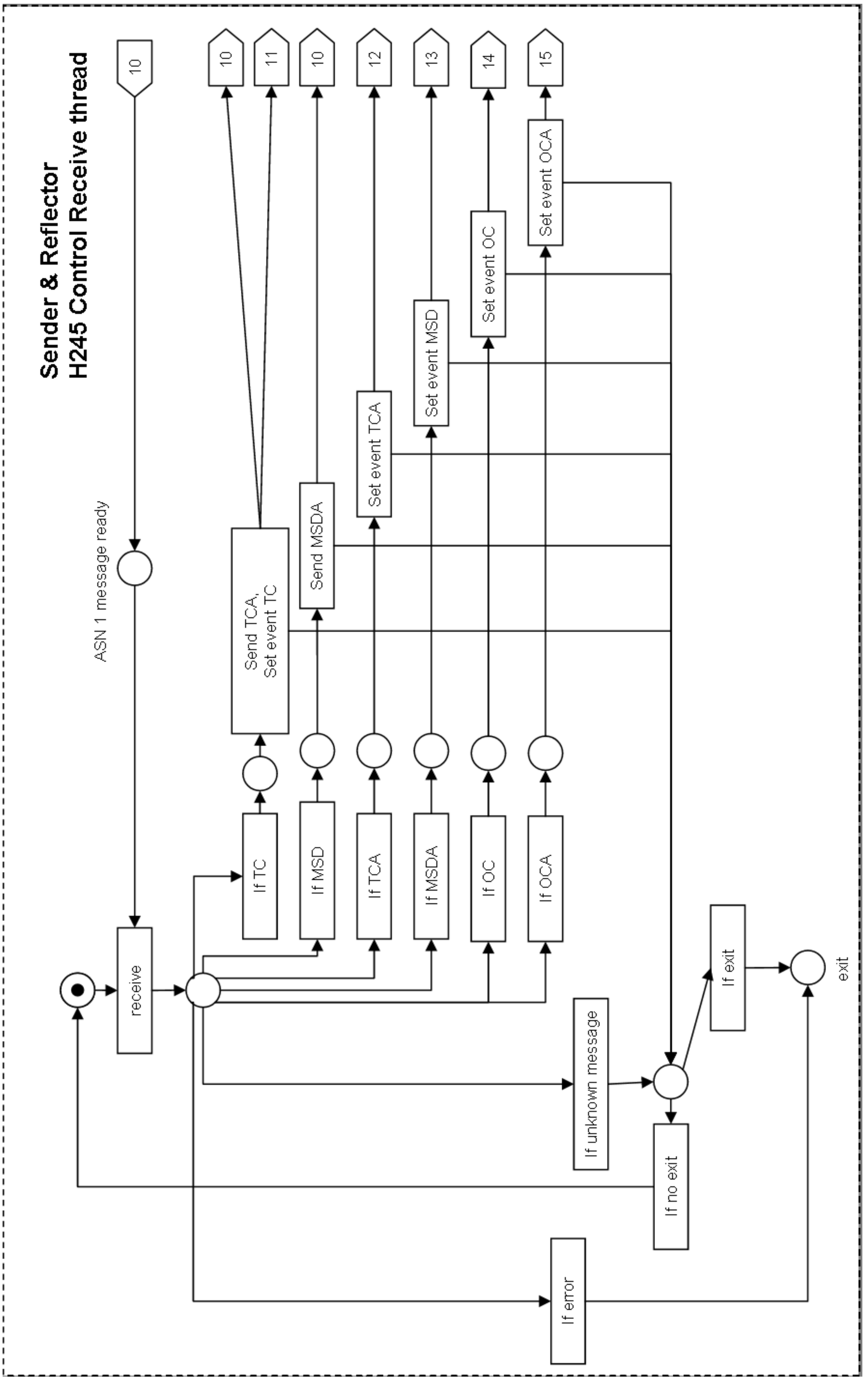
A.6 Call Procedure threads

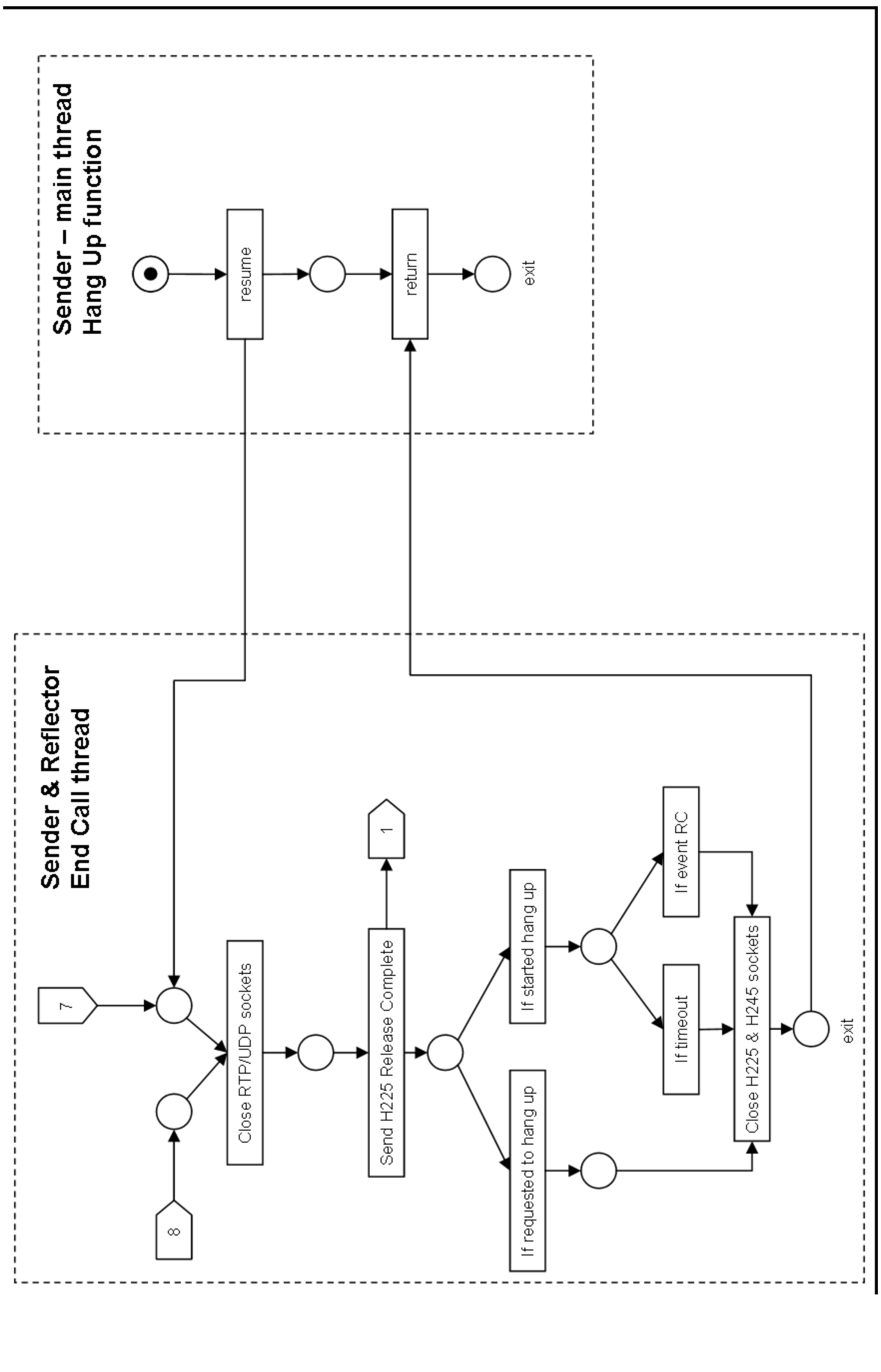
This appendix contains the Petri-Nets diagrams of call procedures threads.


Sender – main thread
Make Call function
Connect to H225
Send Setup
Start timer T1
1
If timeout T1
If event CP & A & C
3
4
5
Connect to H245
9
Send H245 Terminal Capability
Send H245 M/S Determination
Start timer T2
10
If timeout T2
If event TC,TCA,MSD
11
12
13
Send H245 Open Channel
Start timer T3
10
If timeout T3
If event OC
14
15
Start EndCall thread
8
exit


Reflector – Incoming Call thread
2
Start EndCall Thread
8
Send Call Proceeding
Send Alert
Send Connect
Start timer T1
1
9
If timeout T1
If client connected on H245
Send H245 Terminal Capability
Send H245 M/S Determination
Start timer T2
10
11
13
14
If timeout T2
If event TC,TCA,MSD
Send H245 Open Channel
Start timer T3
10
15
16
If timeout T3
If event OC & OCA
exit

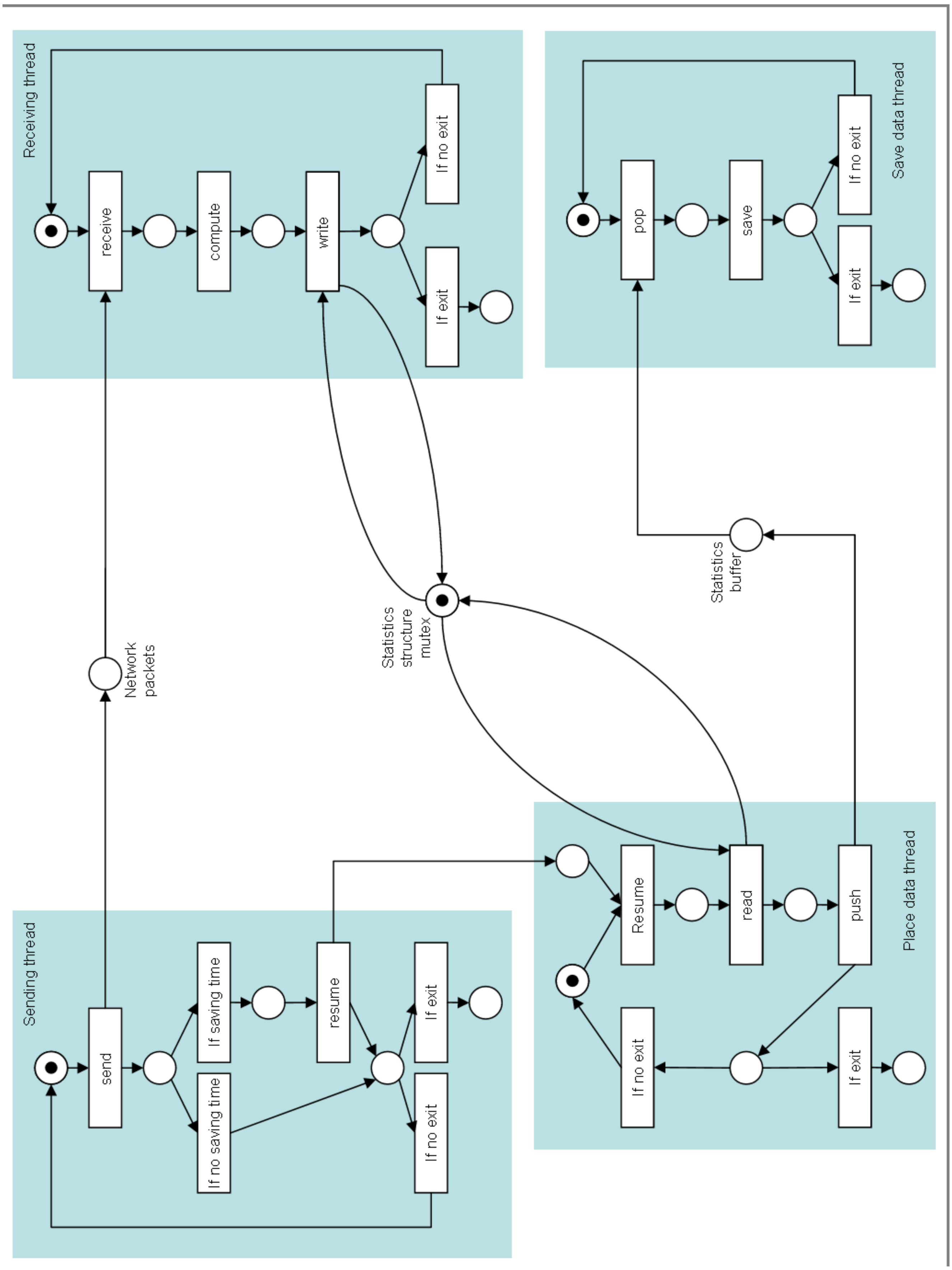

Sender & Reflector
H225 Call Control Receive thread
1
ASN 1 message ready
receive
If Setup
If Call Proceeding
If Alert
If Connect
If Release Complete
Start Incoming Call thread
Set event CP
Set event A
Set event C
Set event RC
resume
2
3
4
5
6
7
If unknown message
If no exit
If exit
exit
If error


Sender & Reflector
H245 Control Receive thread
10
ASN 1 message ready
receive
If TC
If MSD
If TCA
If MSDA
If OC
If OCA
Send TCA,
Set event TC
Send MSDA
Set event TCA
Set event MSD
Set event OC
Set event OCA
10
11
10
12
13
14
15
If unknown message
If exit
If no exit
If error
exit


Sender – main thread
Hang Up function
resume
return
exit
Sender & Reflector
End Call thread
7
8
Close RTP/UDP sockets
Send H225 Release Complete
1
If started hang up
If requested to hang up
If event RC
If timeout
Close H225 & H245 sockets
exit

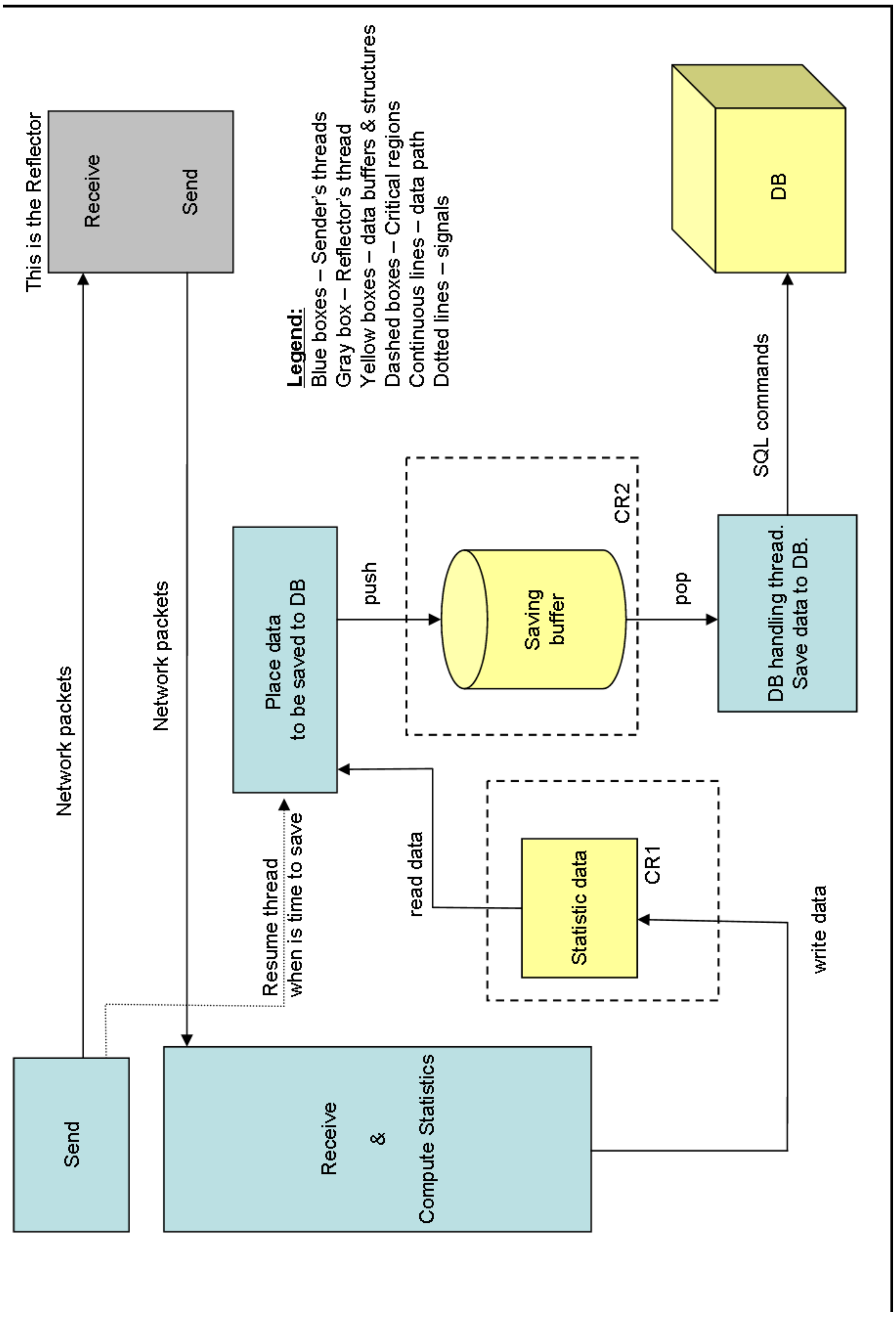
A.7 RTP Packets and Statistic Computation threads

This appendix contains the Petri-Nets diagrams of threads involved in RTP packets transmission and Statistic computation and saving to data-base.


Receiving thread
receive
compute
write
If no exit
If exit
Save data thread
pop
save
If no exit
If exit
Statistics buffer
Statistics structure mutex
Network packets
Sending thread
send
If saving time
If no saving time
resume
If exit
If no exit
Place data thread
Resume
read
push
If no exit
If exit

A.8 Data Flow in Packets Transmission Stage

This appendix contains the diagram of data flow between threads, from sending of network packets up to receiving, computation of Statistics and saving them into the data-base.


This is the Reflector
Receive
Send
Legend:
Blue boxes – Sender's threads
Gray box – Reflector's thread
Yellow boxes – data buffers & structures
Dashed boxes – Critical regions
Continuous lines – data path
Dotted lines – signals
DB
SQL commands
CR2
push
Saving buffer
pop
Place data to be saved to DB
DB handling thread. Save data to DB.
Network packets
Network packets
Resume thread when is time to save
read data
Statistic data
CR1
write data
Send
Receive & Compute Statistics

A.9 Timers Testing Procedure

This appendix describe an algorithm used for measuring the accuracy of some time functions. See also the studies of mr. Chih-Hao Tsai at http://www.geocities.com/hao510/w98timer/.

The following algorithm tries to measure the precision of reporting of the time value of some time functions provided by standard C run-time library and Win32 API, by measuring the actual response and comparing it to the expected one.

The algorith is the following:

```
for (i = 100; i <= 120; i++)
{
    start = time_function();
    while ( time_function () < (start + i));
    finish = time_function () ();
    duration = finish - start;
    printf ("expected:%d actual:%ld ", i, duration);
}
```

The function time_function() used above may be any time function that retrieves the current time value. It returns the time value in time units that may be seconds, milliseconds or in case of other functions like QueryPerformanceCounter, fractions of a second up to microseconds. As may be seen the test is repeated 20 times. Each time the duration of execution of the "while" loop is expected to be as many time units as given by the variable i, that takes values between 100 and 119 (20 values). The actual duration of execution of the "while" loop is computed by the difference between variables "finish" and "start" and kept in variable "duration". A comparison between the variables "duration" and "i" shows the difference between the actual duration and expected duration, showing the precission and actual resolution of the measured time function. The results may look like in the next example (clock() function returning milliseconds).

```
expected:100 actual:100, err: 0.000000 ms
expected:101 actual:110, err: 9.000000 ms
expected:102 actual:111, err: 9.000000 ms
expected:103 actual:110, err: 7.000000 ms
expected:104 actual:110, err: 6.000000 ms
expected:105 actual:110, err: 5.000000 ms
expected:106 actual:110, err: 4.000000 ms
expected:107 actual:110, err: 3.000000 ms
```

```
expected:108 actual:111, err: 3.000000 ms
expected:109 actual:110, err: 1.000000 ms
expected:110 actual:110, err: 0.000000 ms
expected:111 actual:120, err: 9.000000 ms
expected:112 actual:120, err: 8.000000 ms
expected:113 actual:120, err: 7.000000 ms
expected:114 actual:121, err: 7.000000 ms
expected:115 actual:120, err: 5.000000 ms
expected:116 actual:120, err: 4.000000 ms
expected:117 actual:120, err: 3.000000 ms
expected:118 actual:120, err: 2.000000 ms
expected:119 actual:120, err: 1.000000 ms
```

It may be seen that in this case the maximum error is 9 ms so the function is considered to have a precision of around 9 ms.

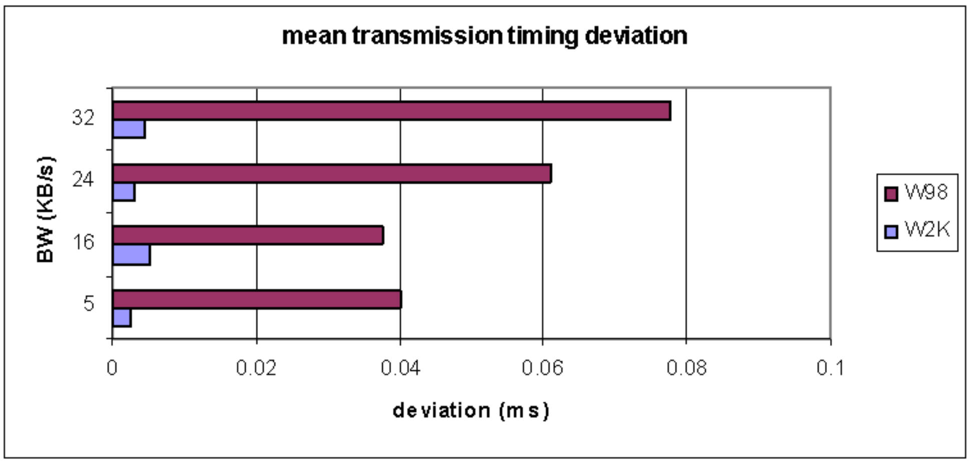
This algorithm may be used to test any other time function. With small modifications it may be used to test also timer services like sleep or SetTimer.

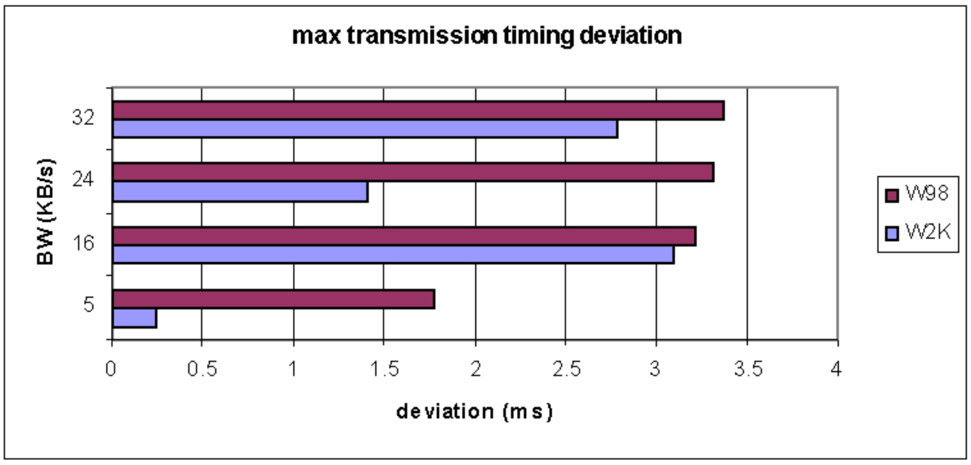
Appendix B

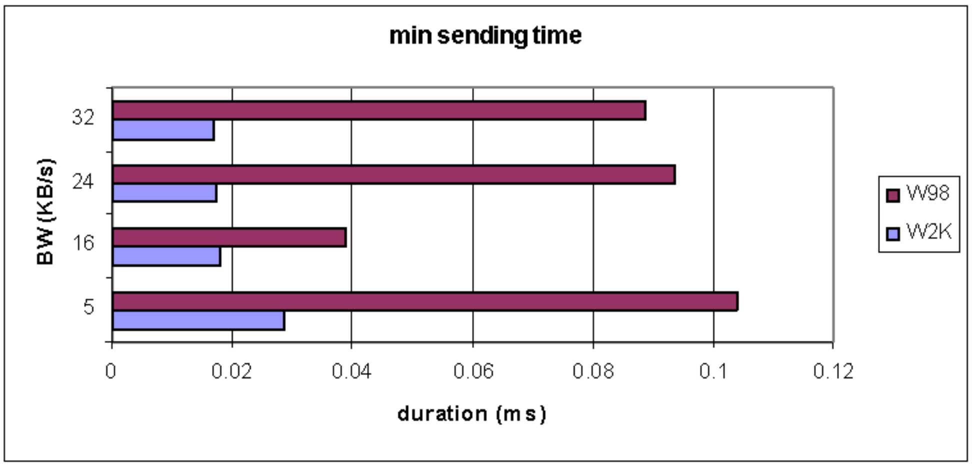
Graphs

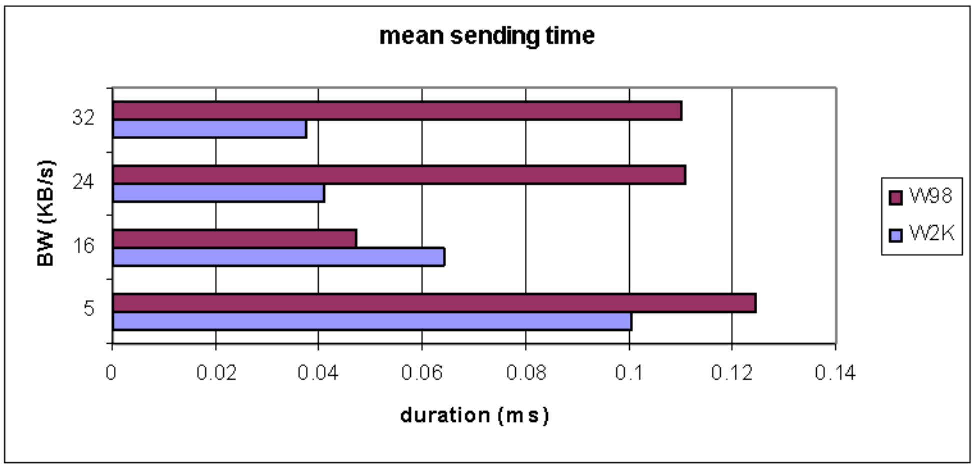
B.1 Transmission Timing Deviation Charts

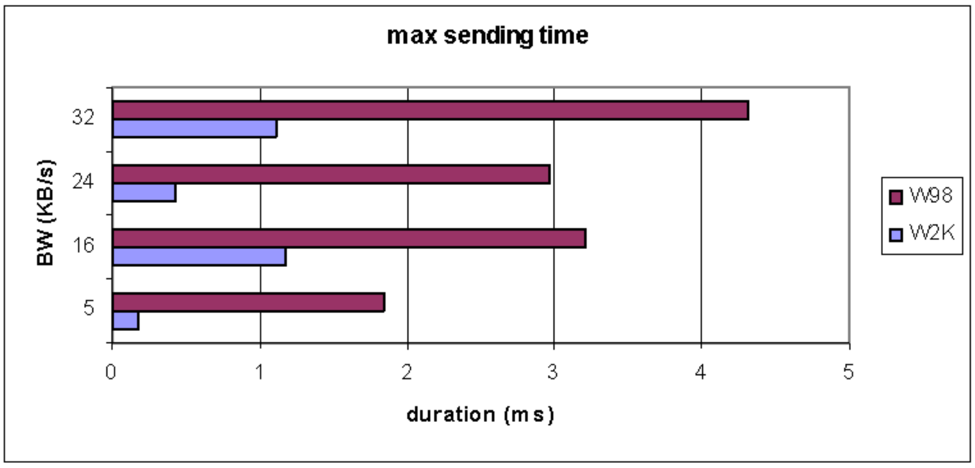
The first two charts show the differences between transmission timing deviations obtained on two different systems: Athlon 1GHz with Win 2K and an Athlon 500MHz with Win 98. The last three charts show the differences between sending function execution time obtained on two different systems: Athlon 1GHz with Win 2K and an Athlon 500MHz with Win 98.


mean transmission timing deviation
BW (KB/s)
32
24
16
5
0
0.02
0.04
0.06
0.08
0.1
deviation (ms)
W98
W2K


max transmission timing deviation
BW (KB/s)
32
24
16
5
0
0.5
1
1.5
2
2.5
3
3.5
4
deviation (ms)
W98
W2K


min sending time
BW (KB/s)
32
24
16
5
0
0.02
0.04
0.06
0.08
0.1
0.12
duration (ms)
W98
W2K


mean sending time
BW (KB/s)
32
24
16
5
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
duration (ms)
W98
W2K


max sending time
BW (KB/s)
32
24
16
5
0
1
2
3
4
5
duration (ms)
W98
W2K

B.2 Real Environment Measurements Results

Results obtained from measurements on DTU's network.

Sender runs on Windows XP.
Reflector runs on Windows 2000.

Total nr of calls: 30
Calls not established: 0
Calls ended correctly: 30
Calls ended abnormally: 0

Skipped frames: 0
Transmission timing deviation, mean: 0.0418 ms, max: 45.9676 ms
Sending time, min: 0.0369 ms, mean: 0.0605 ms, max: 2.6358 ms

Pkt	**L**	**OO**	**J**	**min D**	**mean D**	**max D**	**min I**	**mean I**	**max I**
200	0	0	0.0842	0.7778	1.3380	2.2265	46.5461	49.9999	53.4512
200	0	0	0.0844	0.7789	1.3888	5.8298	46.0193	50.0000	55.0788
200	0	0	0.0856	0.7747	1.4026	7.4725	43.3435	49.9997	55.6101
200	0	0	0.0867	0.7775	1.4515	9.0894	42.7750	50.0001	58.3057
200	0	0	0.1011	0.7803	1.4671	9.1679	41.6441	49.9998	58.3641
200	0	0	0.1015	0.7780	1.4987	12.5396	41.3147	49.9998	58.6424
200	0	0	0.0735	0.7764	1.4968	10.9623	39.7970	50.0052	59.1285
200	0	0	0.0855	0.7778	1.4606	9.0419	42.8163	49.9964	57.3696
200	0	0	0.1729	0.7803	1.4313	9.7845	40.9750	50.0107	57.9317
200	0	0	0.3101	0.7808	1.4124	9.9462	41.9274	50.0127	58.1392
200	0	0	0.2570	0.7853	1.4214	5.6164	46.6165	50.0244	53.7289
200	0	0	0.4808	0.7766	1.4705	7.2115	45.1258	50.0268	56.4156
200	0	0	0.6025	0.7797	1.4873	8.8481	44.2045	50.0404	56.9824
200	0	0	0.7690	0.7797	1.5006	10.4167	42.0905	50.0429	59.6500
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Bibliography

[1] International Telecommunication Union (ITU-T), *H.323 - Packet-Based Multimedia Communication Systems*
URL : http://www.itu.org

[2] International Telecommunication Union (ITU-T), *H225.0 - Call Signaling Protocols and Media Stream Packetization for Packet-Based Multimedia Communication Systems*
URL : http://www.itu.org

[3] International Telecommunication Union (ITU-T), *H.245 - Control Protocol for Multimedia Communication*
URL : http://www.itu.org

[4] International Telecommunication Union (ITU-T), *Q.931 - ISDN user-network interface layer 3 specification for basic call control*
URL : http://www.itu.org

[5] Agilent Technologies, white paper
URL : http://www.agilent.com

[6] CISCO Systems, *Understanding Delay in Packet Voice Networks*, white paper
URL: http://www.cisco.com

[7] Magis Networks Inc. , *Quality of Service Considerations in a Multimedia Home Network*, white Paper Feb. 2001
URL : http://www.magisnetworks.com

[8] Most Valuable Professional - MVPS, *Selecting Timer Functions*, Robert Dunlop
URL : http://www.mvps.org

[9] Most Valuable Professional - MVPS, *Writing the Game Loop*, Robert Dunlop
URL : http://www.mvps.org

[10] Texas Instruments, *Voice over Packet* White paper - Jan. 1998

[11] Microsoft Software Developer Network, *Articles:* Timer Functions, Time Management, Multimedia Timer Functions, Article ID: Q45702