University of Hertfordshire, Hatfield, UK

School of Electronic, Communication and Electrical Engineering

University of Hertfordshire, Hatfield, AL109AB, UK Phone: +44 (0)1707 28400

Improving Telecommunication Performance for Robot Teleguide:

A Study of TCP,RTP and RTSP protocols

Simona D’Asero

Settembre 2009

Supervisors:
Dr. Salvatore Livatino
University of Hertfordshire, United Kingdom

Prof. Giovanni Muscato
University of Catania, Italy

Contents

Chapter 1

Introduction

Application of robotic devices can be significantly extended by teleoperation – controlling them from a remote place. Internet, considering its wide spread and accessibility, is an ideal media for data transmission between a robot and its operator.[1]

Tele-operated robots are devices used to perform different kind of tasks, most of them act in dangerous or hazardous environments. All these kind of tasks have unpredictable and non-repetitive aspects that needs a remote driver.

This project starts from the collaboration among the University of Hertfordshire and the University of Catania which has allowed the realization of the wheeled mobile robot 3MO.R.D.U.C.

The main aim of this project is to study the advantages brought by the use of a real time protocol in a robot teleguide system. Usually, the operator needs a remote access to the robot, via internet connection. This led to a number of problems related to performances, due to the application real-time constraints. That's why a suitable internet communication protocol has to be used in order to achieve a better teleguide service.

The project addresses the study and test of a real time communication protocol applied to telerobotics systems. In fact, in order to improve MORDUC performances, the use of an appropriate transport protocol for data transmission has been considered. This work focuses on the investigation of the RTP protocol, highlighting the efficiency difference compared to the currently used TCP protocol.

Starting evaluating the state of art in the related fields, such as telerobotics systems connected to the Internet and the study of the robotic platform 3MO.R.D.U.C,, the project focuses are:

1. Study of RTP protocol for real time application;

2. Study of RTSP protocol for data streaming;
3. Performing tests on developed application for evaluating protocols performance, providing numerical statistics for the comparison between RTP and TCP;

This work was developed with the support of Prof. S. Livatino who is currently with the School of Electronic, Communication and Electrical Engineering at the University of Hertfordshire, United Kingdom.

1.1.Internet Telerobotics

A robot according to [2] is "any automatically operated machine that replaces human effort, though it may not resemble human beings in appearance or perform functions in a humanlike manner".

Robotics was born as science to develop machines, destined to factories, capable to accomplish boring, repetitive and simple operation as human substitute to product goods in a cheaper way.

A new branch of robotics is gaining importance as we can see in literature***:*** Telerobotics.It is the area of robotics concerned with the control of robots from a distance, mainly using wireless connections or the internet.

The Telerobotics using the Internet as the communication link is a new research area. Several research groups are working in this area mainly due to the low cost access provided by the Internet. The growing interest in this field is stimulated by the advancement of Internet, which provides access to various computing resources virtually from everywhere in the world. Increasing application of Internet as communication media for telerobotics system also comes from the fact that it uses standard communication protocol, and that the physical media is readily available for telerobotics application, eliminating the need for developing a dedicated, proprietary, and expensive communication system.

Telerobotics originated to allow humans to manipulate objects in remote or hostile environments. It allows people to actively participate in remote exploration in various applications. Areas where internet telerobotics is predicted to be useful include entertainment, telemanufacturing, telemedicine, teleoperation and mining. It has tremendous implications for education and training, allowing students and trainers to actively explore remote environments . It also has implications for research as well, where laboratories can share access to expensive resources. There are problems, however, with the limited bandwidth of Internet for telerobotics system, particularly in regards to the visualization of robot movement from remote site, and with the security of Internet.

1.2.Networks and Multimedia

In according to [4] Internet was born from ArpaNet ('60), a network created by the U.S. Army. The ArpaNet aim was to build a decentralized network to be high fault tolerant, avoiding problems during possible attacks. The first internet, as it is today, was a network among several U.S. colleges using ArpaNet facilities. Since the 1992 the internet grown every year in user as in services. Nowadays instruments like web and emails are used as one of the best way to exchange informations.

The global network met the multimedia in 1991 thanks to Tim Berners-Lee, he was the father of HTTP, a protocol which permit to read material in a non sequential way (hypertext). A protocol is a grammar to permit different computers, through software, to communicate among them. HTTP, POP, IMAP, SMTP, FTP, DNS are the most known by users.

Images and sounds are important in every day life, and early they met the network increasing the power of communication of internet.Since those days the use of networks to send images, video and music increased and most of the traffic, today, is through websites which stream video, such as YouTube, video conference and instant messaging tools like Skype.

Multimedia is different from text. A text, binary or not, have to be transmitted without loosing data during the transmission because it will mean receive something unuseful. Multimedia characteristics, in the other hand, pointed out the need of particular protocols to improve the throughtput, because human kind nature is not to sensible to noise or data loss in images, videos or music. To handle the intrinsic multimedia's real time properties we talk about RTP, MMS and other closed source protocols they use a similar approach to mp3, jpeg and mpeg. They are three lossy file formats, because they discard some data, losing in quality, using compression algorithms to decrease the dimension of files but they are still enjoyable

1.3 Core idea and argumentation

The main purpose of this project is to investigate benefits of use RTP protocol for real time video and data transmission in a teleoperated mobile robot system and in particular in the 3MORDUC robot.

This project is the first step to rewrite or modify properly the existent MORDUC client server communication making it to be more efficient and providing better performances for the camera, laser and sensors data exchanged.

Chapter 2
Background knowledge

This chapter is aimed at providing basic background information and concepts, that the reader needs in order to understand the proposed method. In this chapter are introduced the result of state of the art research and streaming protocols.

2.1 State of the art

There have been many web telerobotics experiments, some of which are briefly discussed below before to start with the project development.

In this paper [1] the authors have designed and implemented an experimental system to test and evaluate the performance and usability of a modern web telerobot architecture. They have used the RTP protocols protocol (in UDP packets) to transmit time-sensitive video data and control signals. Video data use the Java Media Framework to transfer the images with JPEG or H263 coding. Commands are transmitted with TCP because reliable transmission is a requirement, and other administrative and not time-sensitive data are also use TCP sockets.

According to the results of experiments with used system the authors conclude that current Internet and web technology enables teleoperation of robotic devices through WWW, though the low quality of Internet services requires the use of special real-time protocols and compression.

Fiorini and Oboe[3] investigated the behaviour of the Internet with a view to continuous control of a telerobot using force feedback. Results of network performance tests between fixed nodes over the Internet suggested that the delay characteristics of a link can be modelled using the mean and variance of the round trip delay. They also identified a second problem not previously considered in traditional telerobotics; that of packet loss. A protocol that considers both these factors and provides some form of guaranteed performance is defined as a Real Time Network (RTN) protocol: Real Time Network protocols are designed to connect clients with specific performance requirements, and to guarantee the fulfilment of those requirements. The performance is intended as desired throughput, delay and reliability. They discuss two approaches to providing RTN services; either to use existing standards at the cost of accountability, or to emphasise guaranteed performance at the cost of compatibility. The real time protocol (RTP) with its associated real time control protocol (RTCP) is an example of the first type of protocol and is used for multimedia streaming over the Internet. *Tenet* is an example of the second type, and was designed on the premise that no RTN can be built over a data link which does not guarantee a maximum delivery time (IP does not guarantee a maximum delivery time). Tenet therefore only works on network architectures that provide this guarantee such as FDDI and ATM 2. Finally, they recommend the most viable approach for real-time control is to base development on RTP.

The paper [5] presented the development of the RobWebLink robotic teleoperation system. The implemented system is based on a client/server architecture using a UNIX platform (Linux) as the HTTP server, which provides the WWW service. The client accesses the robot through the telerobotic server (WebRobot) by sending WWW forms and receiving the robot telegrams as answers. The application protocol to permit server and client to communicate in order to stream video and sending parameters has been implemented using the TCP/IP protocol. In this work good results are obtained relative to the implementation of a through−the−internet communication link between the robot controller and a remote WWW client. But to guarantee the real time operation of the system for future works the authors propose the use of RTP protocol.

2.1 Network and Multimedia

2.1.1 Network Protocols

The TCP/IP model or Internet reference model describe the communication between two or more computers, using networks, through a stack of 4 layers:

- **Network Access Layer:** it describe specifications about all the electrical and physical characteristics of network's devices, it provides specifications to transfer data between network entities, and correct possible errors at physical level.

- **Internet or Internetworking Layer**: it provides the functional and procedural means of transferring variable length data sequences from a source to a destination, taking care about the quality of service.

- **Transport Layer** : it provides transparent data transfer between end users, through flow control, segmentation/desegmentation, and error control it may take care about reliability (TCP and UDP).

- **Application Layer**: This is where the high level protocols live and operate.

We will focus on the Transport and Application Layer, in fact we want to develop a language in a client server application that exchange data.

Since networking was born, and in particular internet, many kind of application protocols where designed, optimized to handle different kind of data.
Considering multimedia data, we can focus our attention over the following well known protocols:

- Real Time Family: RTP (Real Time Protocol), RTCP (Real time control protocol),
- RTSP (Real Time Stream Protocol)

2.1.2 Protocols Designed for Multimedia

2.1.2.1 Real-time transport protocol (RTP)

The Network Working Group designed a protocol that provides end-to-end network transport functions suitable for applications transmitting real-time data: the Real-Time Transport Protocol (RTP).

RTP provides end-to-end transport functions which are suitable for applications transmitting real-time data, such as digital audio/video over multicast or unicast networks [1] but it is also applicable for other data types used in teleoperation.

These functions include: content identification of payload data, sequence numbering, timestamping, and monitoring QoS of data transmission.

RTP consists of two protocols: RTP for real-time transmission of data packets and RTCP for monitoring QoS and for conveying participants' identities in a session. When referring to RTP, we mean both RTP and RTCP if not otherwise explicitly stated. RTP is integrated within the application. Its packets are usually encapsulated in UDP packets which provides checksumming and multiplexing. RTP uses mainly IP multicasting for distribution of packets within a multicast RTP session.

RTP data packets consist of a header followed by payload data which can be either a video frame or several audio samples [6].

Some fields in a RTP header are:

- *Payload type:* identifies the format of RTP payload, for example H.261 for video streams.
- *Sequence number:* is incremented by one for each data packet sent. It can be used by the receivers to detect packet loss and out-of-sequence packets.
- *Timestamp:* is the instant the data packet is generated. It is used in synchronisation and jitter calculation.

RTCP is used in conjunction with RTP to monitor the quality of service and to convey information about the participants in an on-going session. Each participant in RTP session periodically transmits RTCP control packets to all other participants, in that way feedback messages can be used to control performances, the sender has the ability to modify its transmissions rules depending on the feedback received.

Each RTCP packet contains sender and/or receiver reports with statistics including number of packets sent, number of packets lost, interarrival jitter, etc.

In order to improve the performances of MORDUC it is necessary to use real time transport protocol for transmit the data. For this reason we have decided to use the RTP protocol to developed client server application.

2.1.2.2 The Real-Time Streaming Protocol (RTSP)

RTSP was developed by the Multiparty Multimedia Session Control Working Group (MMUSIC WG) of the Internet Engineering Task-Force (IETF) and published as RFC 2326 in 1998. It is an application-level protocol for control over the delivery of data with real-time properties.

RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. The streams controlled by RTSP may use RTP, but the operation of RTSP does not depend on the transport mechanism used to carry continuous media.

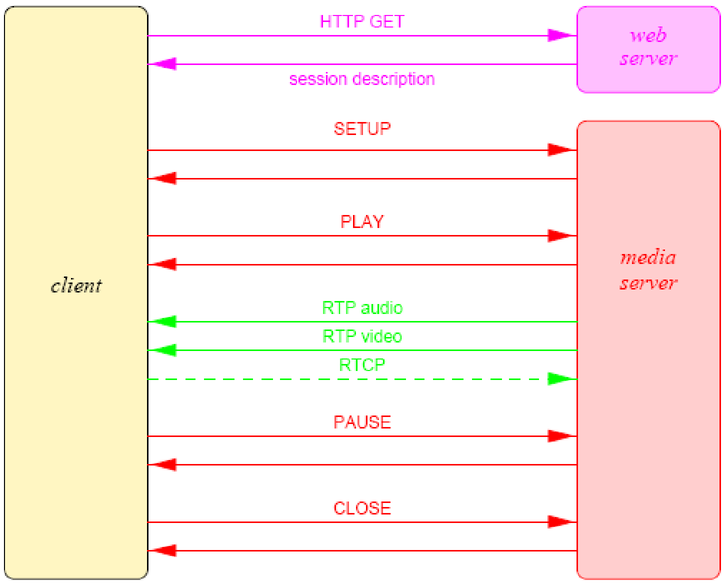
The RTSP protocol is intentionally similar in syntax and operation to HTTP/1.1. [8]. HTTP extension mechanisms can in most cases also be added to RTSP. However, RTSP differs in a number of important aspects from HTTP.

HTTP is an asymmetric protocol where the client issues requests and the server responds. In RTSP, both the media client and media server can issue requests. While HTTP is stateless, RTSP is a stateful protocol.There is no notion of an RTSP connection. Rather, an RTSP server maintains a session labeled by an identifier. A session identifier is used to keep track of sessions when needed. RTSP messages are sent from client to server, although some exceptions exist where the server will send to the client.

Many methods in RTSP do not contribute to state. However, the following play a central role in defining the allocation and usage of stream resources on the server: SETUP, PLAY, RECORD, PAUSE, and TEARDOWN.

- SETUP: Causes the server to allocate resources for a stream and start an RTSP session.

- PLAY and RECORD: Starts data transmission on a stream allocated via SETUP.
- PAUSE: Temporarily halts a stream without freeing server resources.
- TEARDOWN: It is used to terminate the session. It stops all media streams and frees all session related data on the server



The SETUP and TEARDOWN (or CLOSE) methods provide a proxy with all of the port (num:554) information necessary to open up, map and close ports. The client sends the SETUP method to the server. The data in the SETUP method specifies the set of delivery transports the client can handle.

The client sends the TEARDOWN method to the server to stop media data delivery. When the TEARDOWN method is received, the ports associated with this session can now be deallocated.

In a future works it will possible use RTSP to handle commands between client and server MORDUC.

Chapter 3

Previous work in Mobile Robot Teleguide

The Chapter 3 is aimed at providing basic information and concepts about the robot hardware- software architecture to well understand the project's scenario.

2.2 The 3MORDUC robot

The 3MO.R.D.U.C., "3rd version of the MObile Robot DIEES University of Catania", shown in Figure 1., is a wheeled mobile robot in differential drive configuration. This open robotic platform is successfully used in localization and navigation experiments.

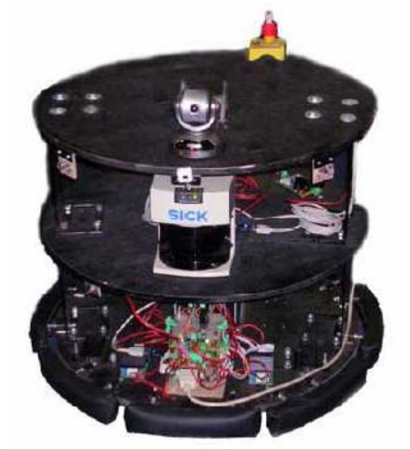


Fig.1.

The movement is accomplished by two 40W DC motors, Maxon F2260, and the motor axes is linked with a gear box (gear ratio 1:19). Two rubber wheels are linked with the gear box axis and a third castor wheel is free to rotate, facilitating so the execution of the curves.

The robot structure as shown in figures 1 and 2 has three shelve linked together.

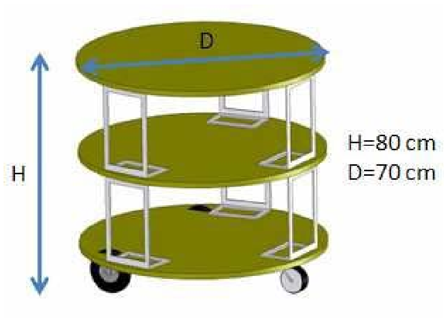


Fig.2

On the lower shelf there are two lead batteries (12V/18Ah) which provide the power supply. The robot autonomy is about 30 minutes for continuous working.

An on board electronic rack controls each module of the robot (motion, sensors and communication). On the robot there are several sensors monitoring the workspace and the robot state (bumper, encoder, laser, sonar and stereocam).

Finally on the top of the robot there is a laptop where is placed the application for the robot control. There are also: a joystick for the manual handling of the robot and an emergency button.

In the next section will be presented the main features of all sensors present on the robot.

2.3 Analysis of data sensors

Before to start with the client server application development, it was important to well understand the different sensor solutions and what kind of data the robot Morduc manages, in particular their type and format. (i.e. integer/float numbers , precision).

The sensors on board are analyzed in the following section:

- **Bumpers**

A belt of Bumpers around the entire perimeter of the robot is mounted on the base, over the wheel. These sensors have to recognize and reduce damages in a collision. The bumpers are simple switches pushed when there is a collision. They are connected to the same bus of the sonar sensors (I2C).

- **Encoders**

A digital optical encoder is a device that converts motion into a sequence of digital pulses. The pulses can be converted to relative or absolute position measurements, by counting a single bit or by decoding a set of bits. On the robot there are placed two incremental encoders with a resolution of 500 pulses/turn.

- **Sonar SRF08**

The sonar sensors measure the distance from an obstacle using the flight time of an ultrasonic signal produced by means of a vibrating piezoelectric sensor. Sonar Sensor used on the robot are 8 SRF08 connected to the bus I2C. In this kind of sensors there is, also, a photo-resistor that allows the sensing of the environmental brightness.

- **Laser Sick LMS 200**

The LMS200 operates by measuring the time of flight of laser light pulses: a pulsed laser beam is emitted and reflected if it meets an object. The reflection is registered by the LMS200's receiver. The time between transmission and reception of the impulse is directly proportional to the distance between the LMS200 and the object (time of flight). In a radial field of vision, a light impulse (spot) is emitted every 0.25°, 0.5° or 1° (depending on the set variant).[9]

In the MORDUC, angular resolution, scanning angle and number of measured values are:

Angular resolution =1°
Max. scanning angle =180°
Max. no. of measured values =181
Scanning speed or Response Time: 13.3 ms per revolution

In fact during the communication it is possible read 181 integer to form an array.

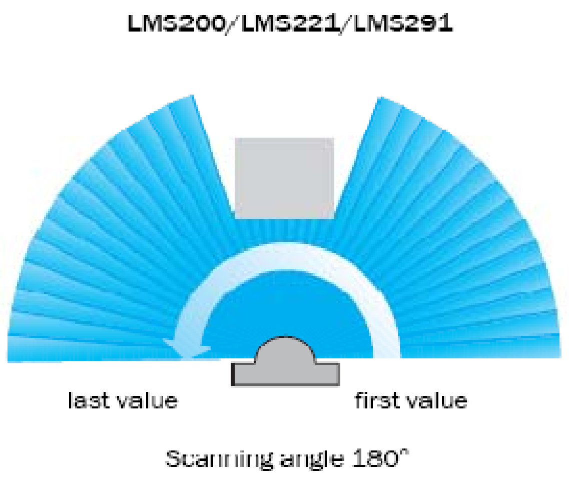


Fig. 3

Figure 3 shows the direction of transmission and maximum scanning angle (standard devices) on top view of the devices.

❖ **Stereo Camera**

MORDUC has a STH-MDCS2-VAR Stereo camera. It consists of two high quality stereo cameras which have 1.3 megapixel resolutions. They have 4.0 mm fixed focus lenses. CMOS sensors of these cameras have a good noise immunity and sensibility; moreover, it is possible to adjust all the image parameter, e.g. exposure gain, frame rate, resolution. The cameras are mounted on a rigid support; it permits to simply adjust the baseline in a range 5-20 cm. The images come from the two cameras are synchronized with an 8 KHz clock, generated by using IEEE1394 interface.

Camera Model:	STH-MDCS2-VAR
Video formats @ Frame rate:	High frame rates – 30 Hz for 640x480, 7.5 Hz for 1280x960
Sensor Specifications:	CMOS
Connection:	IEEE 1394 interface to standard PC hardware – carries power and commands to device, data to PC
Variable baseline:	5 – 20 cm
Focal Length:	4.0 mm
Horizantal Angle of view (1/2” format)	~51^{o}

The network connection allows sending the images or sensor data coming from the robot and the movement commands.

In particular the data exchange between the server MORDUC and the web client are:

1. jpeg images of 1280x480 pixels that are two images of 640x480 provided by the left and right on-board camera
2. jpeg images of 200x200 pixels provided by the laser
3. odometry data

The odometry data are:

- ❖ Time in milliseconds: Integer
- ❖ X,Y, Theta (X e Y represent the abscissa and the ordinate in a Cartesian coordinate system in meters, Theta is the angle of rotation): Double.
- ❖ The number of Collisions: Integer
- ❖ The minimum distance from the obstacles: Integer

2.4 The 3MORDUC Network

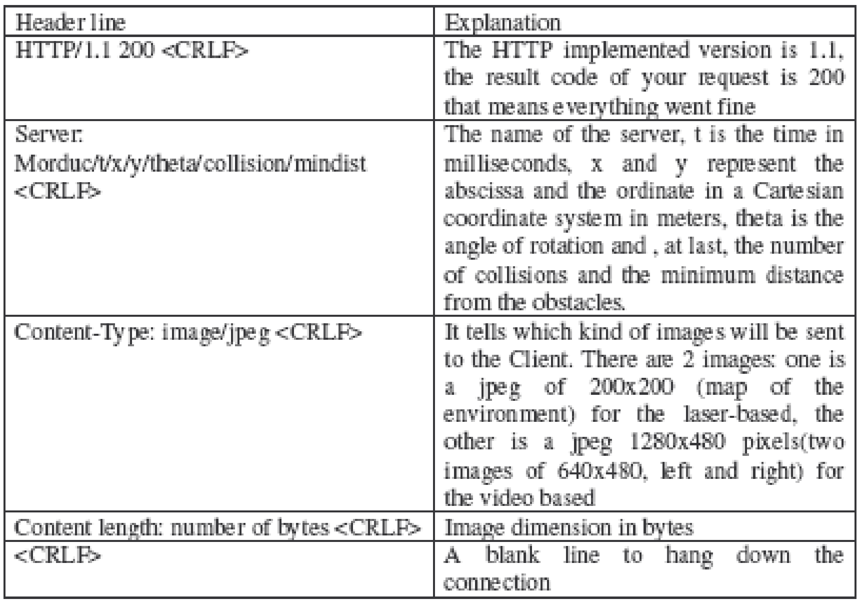
The network system implemented on the 3MORDUC is a typical network client-server architecture. The 3MORDUC software architecture was made to work over the Internet: the client application uses the TCP/IP socket on the standard port 80 (HTTP protocol) to communicate with the server.

The Server was developed in Borland Delphi 7, an object oriented language derived by Pascal. Several classes are wrapper of the windows A.P.I. making really simple to develop efficient code in fast way. The choice of this programming language come from the necessity to include it as part of the control system designed in Catania for the 3 MO.R.D.U.C. robot.

The Client was developed using MFC, Microsoft Foundation Classes in Visual C++ (Visual Studio 2005). It uses OpenGL libraries to create 3D synthetic images and to handle the different kinds of used VR instruments. For a specific study view the [11].

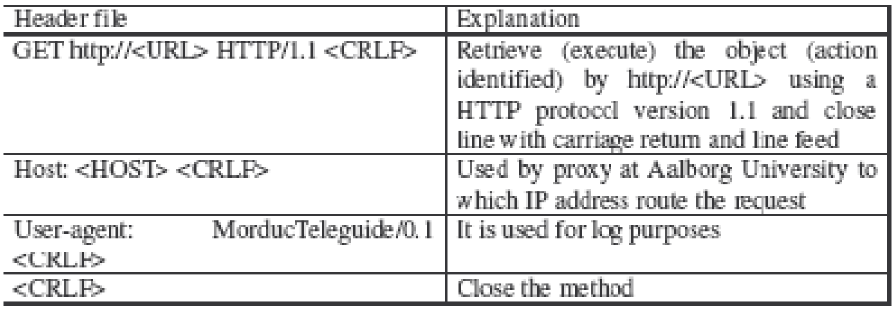
For our project is important to talk about thread communication between client and server. The operations necessary for the communication are related to the socket, such as: socket creation and socket opening.

If there is the server connection, it is allowed to send and receive requests to/from the server. The server sends an acknowledge for the request reply that has the following command pattern:



Header line	Explanation
HTTP/1.1 200 <CRLF>	The HTTP implemented version is 1.1, the result code of your request is 200 that means everything went fine
Server: Morduc/t/x/y/theta/collision/mindist <CRLF>	The name of the server, t is the time in milliseconds, x and y represent the abscissa and the ordinate in a Cartesian coordinate system in meters, theta is the angle of rotation and , at last, the number of collisions and the minimum distance from the obstacles.
Content-Type: image/jpeg <CRLF>	It tells which kind of images will be sent to the Client. There are 2 images: one is a jpeg of 200x200 (map of the environment) for the laser-based, the other is a jpeg 1280x480 pixels(two images of 640x480, left and right) for the video based
Content length: number of bytes <CRLF>	Image dimension in bytes
<CRLF>	A blank line to hang down the connection

From the Client side, the GET method to retrieve whatever information is identified by a Request URI (Uniform Resource Identifier). The GET method, as described, has many options but we will use only 4 lines that are fundamental in our use:



Header file	Explanation
GET http://<URL> HTTP/1.1 <CRLF>	Retrieve (execute) the object (action identified) by http://<URL> using a HTTP protocol version 1.1 and close line with carriage return and line feed
Host: <HOST> <CRLF>	Used by proxy at Aalborg University to which IP address route the request
User-agent: MorducTeleguide/0.1 <CRLF>	It is used for log purposes
<CRLF>	Close the method

The GET method is the way to retrieve images and to send commands to the robot.
In particular, the Client implements three families of commands:

1. *image.cmd.jpg (or image.cmd.bmp):* it could be of four types:
   - *stereo.fow.jpg or laser.fow.bmp*: to fetch webcams images or laser image and to move ahead;
   - *stereo.bak.jpg or laser.bak.bmp:* to fetch webcams images or laser image and to move back;
   - *stereo.rgh.jpg or laser.rgh.bmp:* to fetch webcams images or laser image and to turn right;
   - *stereo.lft.jpg or laser.lft.bmp:* to fetch webcams images or laser image and to turn left.
2. *image.jpg (or image.bmp):* it used to ask for images fetched by webcams or laser scanner when the robot does not have to move;
3. *command.how:* where *command* is which action has to be performed and *how* say in which direction, it uses this schema:
   - *step.fow*: to go ahead;
   - *step.bak:* to go back;
   - *turn.rgh:* to turn right;
   - *turn.lft:* to turn left.

Chapter 4

Tests

In the Chapter 4 we will explain the application used for the tests performances and the results analysis

4.1 Application Description

Before to start with the tests we describe the client-server applications used to make the statistics analysis.

4.1.1 RTP event driven client-server application

This client-server application exchanges simple string data through the RTP protocol. It uses the Jrtplib C++ [12] framework which implements all the RTP protocol features described in RFC 3550 [6] . Client and server has to establish an RTP session before communicating to each other, this session has just one active transmitter (server) and a single receiver (client). Due to the RTP protocol behaviour, the server session has to be configured in order to add the client IP as a transmission destination. A portbase (for RTCP protocol) and a estimation port have to be added too. On the other side, the only thing the client must know is the port through which it can receive server data.
The application allow the user to set a number of parameters:

- Destination address (IP and port)
- Session portbase
- Simulation time (i.e. how many seconds the server has to send data to the client), used to compute the throughput.
- Verbose mode (i.e. either the application has to print status messages on the screen or not).

Once the session has been created, the server starts sending RTP packets to the client until the simulation time has finished. When time expires, it shows the number of packets and bytes sent to the client.

During the simulation, the server sends to the client an “Hello client” string followed by a progressive number as a string buffer with a 128 bytes fixed size.

4.1.2 TCP synchronous client-server application

The applications allow the user to set the following parameter:

- Server connection IP and port (client side)
- Simulation time [s]
- Verbose mode
- Synchronous or Asynchronous mode.

Using the synchronous mode, the applications work with a request – reply behaviour, i.e. the server doesn't send anything to the client, always waiting for a request before transmitting.

This approach allows to avoid client congestion while receiving a large amount of data, making the server transmitting at the client speed. Obviously this produces very low throughput (but no packet loss) at client side.

The data exchanged between client and server are the same as the RTP application, so that we can provide a good comparison between the two protocols.

4.1.3 TCP asynchronous client-server application

This client-server application works in the same way as the previous one, but it has an asynchronous behaviour. Once the connection between the client and the server has been established, the server continuously sends TCP packets to the client until the simulation time expires. This approach is the most similar one to the RTP event driven approach, allowing the server to send data without waiting for the client request.

The direct consequence of this choice is that the client congests and loses some server packets, but the provided throughput is so much better than the previous one.

4.1.4 RTP video streaming client-server application

This client-server application provides a video streaming using the Jrtplib for RTP protocol and the OpenCV frameworks (for camera capture). The RTP server use the OpenCV framework to obtain a video capture and show it through a proper window.

The camera resolution is set so that the server can send a whole frame in a single RTP packet (160x120 pixels). This is required because the RTP protocol uses the UDP transport protocol which has a frame size limit of 65535 bytes.

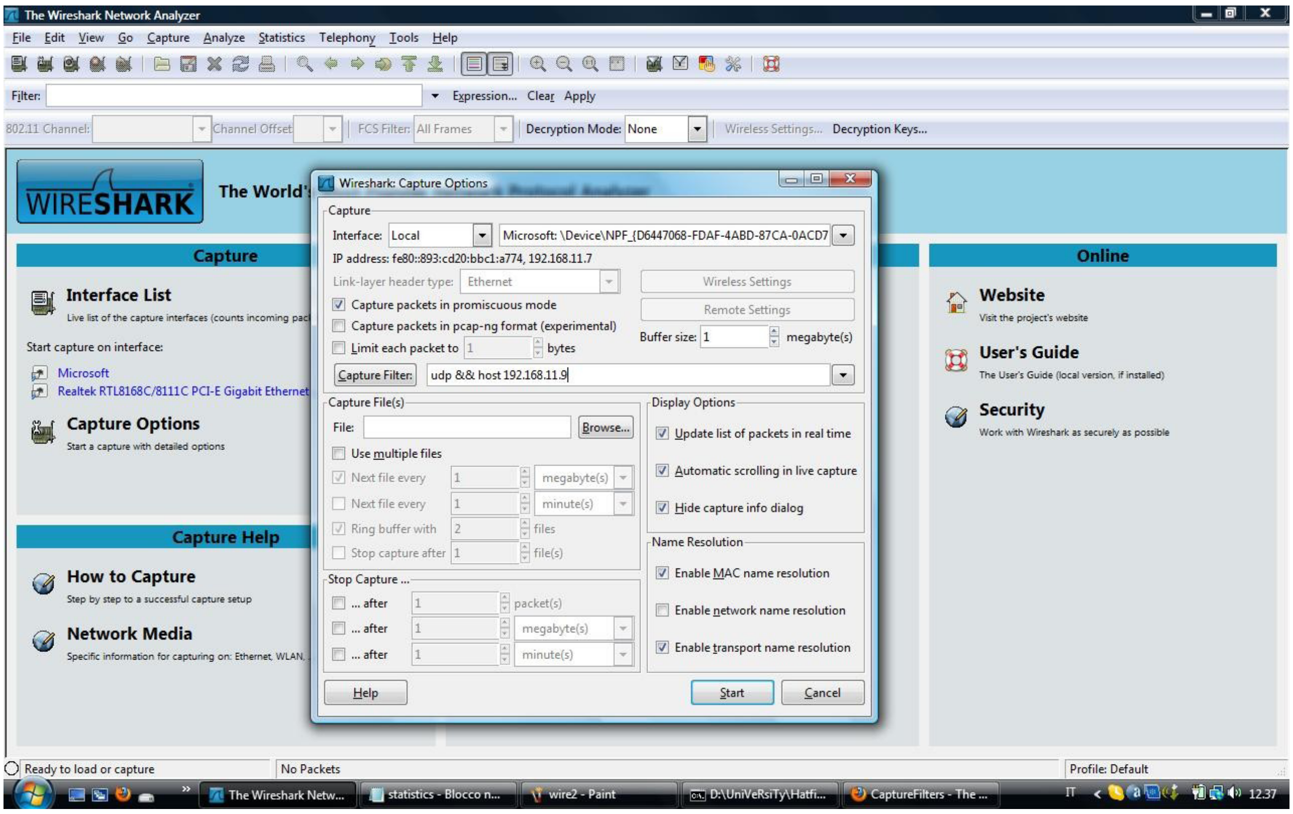
4.1.5 TCP video streaming client-server application

This client-server application provides a video streaming using the TCP protocol and the OpenCV frameworks. In this case the TCP protocol is used in asynchronous mode.

4.2 Tests methodology

4.2.1 Throughput and interarrival jitter evaluation

In order to obtain a quantitative comparison between TCP and RTP protocols, the used client-server applications have been run on LAN network connection, running client and server on different computers connected via ad-hoc 802.11 wireless network and on Internet.


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: Expression... Clear Apply
802.11 Channel: Channel Offset: FCS Filter: All Frames Decryption Mode: None Wireless Settings... Decryption Keys...
WIRESHARK The World's
Capture
Interface List
Live list of the capture interfaces (counts incoming packets)
Start capture on interface:
Microsoft
Realtek RTL8168C/8111C PCI-E Gigabit Ethernet
Capture Options
Start a capture with detailed options
Capture Help
How to Capture
Step by step to a successful capture setup
Network Media
Specific information for capturing on: Ethernet, WLAN, ...
Online
Website
Visit the project's website
User's Guide
The User's Guide (local version, if installed)
Security
Work with Wireshark as securely as possible
Ready to load or capture
No Packets
Profile: Default
The Wireshark Netw...
statistics - Blocco n...
wire2 - Paint
D:\UniVeRsiTy\Hatfi...
CaptureFilters - The ...
IT
12.37

Wireshark: Capture Options

Capture
Interface: Local   Microsoft: \Device\NPF_{D6447068-FDAF-4ABD-87CA-0ACD7
IP address: fe80::893:cd20:bbc1:a774, 192.168.11.7
Link-layer header type: Ethernet   Wireless Settings
☑ Capture packets in promiscuous mode   Remote Settings
☐ Capture packets in pcap-ng format (experimental)   Buffer size: 1 megabyte(s)
☐ Limit each packet to 1 bytes
Capture Filter: udp && host 192.168.11.9

Capture File(s)
File: Browse...
☐ Use multiple files
☑ Next file every 1 megabyte(s)
☐ Next file every 1 minute(s)
☑ Ring buffer with 2 files
☐ Stop capture after 1 file(s)

Stop Capture ...
☐ ... after 1 packet(s)
☐ ... after 1 megabyte(s)
☐ ... after 1 minute(s)

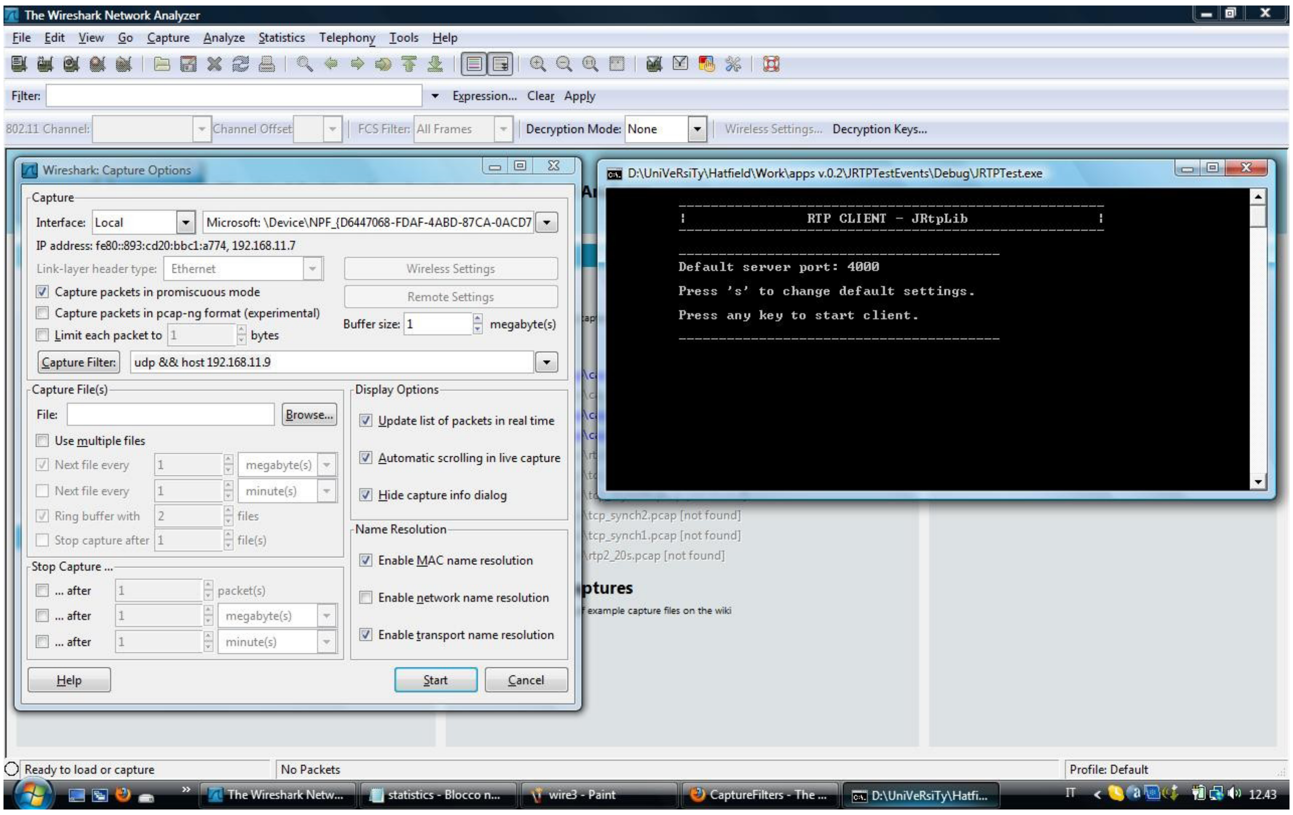
Display Options
☑ Update list of packets in real time
☑ Automatic scrolling in live capture
☑ Hide capture info dialog

Name Resolution
☑ Enable MAC name resolution
☐ Enable network name resolution
☑ Enable transport name resolution

Help   Start   Cancel

Fig 4: WireShark capturing filter for RTP packets only

In addition to the statistics provided by the applications, a sniffing software called WireShark [16] has been used during the simulations to capture transmitted and received packets for a subsequent inspection. WireShark provides also a number of statistics (e.g. Packets received per second) which can be used for a comparison with the ones given by the applications.



```
D:\UniVeRsiTy\Hatfield\Work\apps v.0.2\JRTPTestEvents\Debug\JRTPTest.exe
 ------------------------------------------------------------------
 !                    RTP CLIENT - JRtpLib                        !
 ------------------------------------------------------------------
 -------------------------------------------------
 Default server port: 4000
 Press 's' to change default settings.
 Press any key to start client.
 -------------------------------------------------
```

Wireshark: Capture Options

Capture
Interface: Local — Microsoft: \Device\NPF_{D6447068-FDAF-4ABD-87CA-0ACD7
IP address: fe80::893:cd20:bbc1:a774, 192.168.11.7
Link-layer header type: Ethernet
[✓] Capture packets in promiscuous mode
[ ] Capture packets in pcap-ng format (experimental)
[ ] Limit each packet to 1 bytes
Buffer size: 1 megabyte(s)
Capture Filter: udp && host 192.168.11.9

Display Options
[✓] Update list of packets in real time
[✓] Automatic scrolling in live capture
[✓] Hide capture info dialog

Name Resolution
[✓] Enable MAC name resolution
[ ] Enable network name resolution
[✓] Enable transport name resolution

Ready to load or capture | No Packets | Profile: Default

Figure 5: Running client application and starting capture with WireShark

The tests have been done on the following three cases:

1. RTP traffic
2. TCP asynchronous traffic
3. TCP synchronous traffic

Each of them has been tested performing 15 applications runs, in Local Network (client and server reside on different machine, but remain in the same LAN) and in Internet (client and server are in the different LANs) providing the following statistical information:

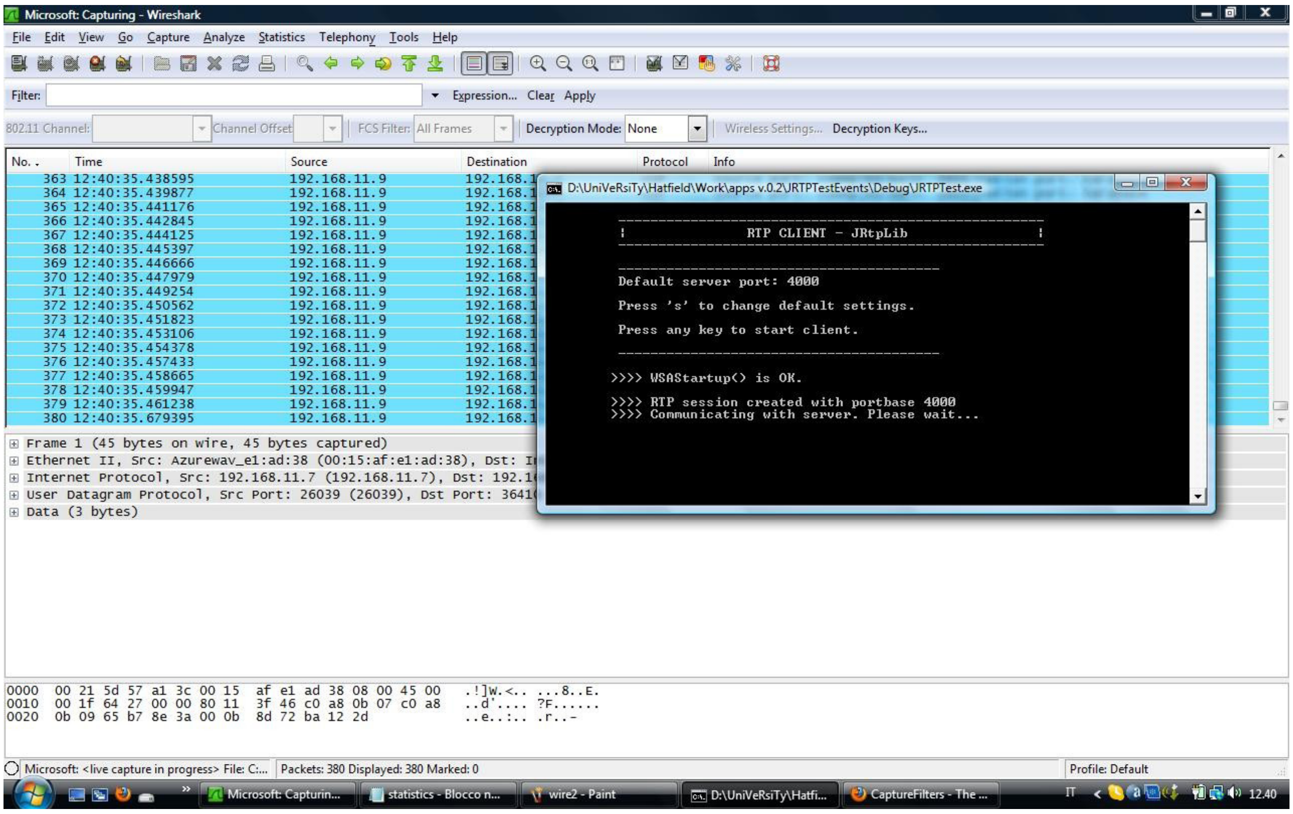
- For TCP applications:

➢ Number of bytes sent by the server
➢ Number of bytes received from the client
➢ Throughput (bytes received / simulation time) [KB/s]

- For RTP application:

  - Number of bytes and packets sent by the server
  - Number of bytes and packets received from the client
  - Throughput [KB/s]
  - Interarrival Jitter [s] [14][15]

Each application run has been performed using a simulation time of 30 seconds.



Decryption Mode: None

```
No. .  Time              Source           Destination   Protocol  Info
  363 12:40:35.438595    192.168.11.9     192.168.1
  364 12:40:35.439877    192.168.11.9     192.168.1
  365 12:40:35.441176    192.168.11.9     192.168.1
  366 12:40:35.442845    192.168.11.9     192.168.1
  367 12:40:35.444125    192.168.11.9     192.168.1
  368 12:40:35.445397    192.168.11.9     192.168.1
  369 12:40:35.446666    192.168.11.9     192.168.1
  370 12:40:35.447979    192.168.11.9     192.168.1
  371 12:40:35.449254    192.168.11.9     192.168.1
  372 12:40:35.450562    192.168.11.9     192.168.1
  373 12:40:35.451823    192.168.11.9     192.168.1
  374 12:40:35.453106    192.168.11.9     192.168.1
  375 12:40:35.454378    192.168.11.9     192.168.1
  376 12:40:35.457433    192.168.11.9     192.168.1
  377 12:40:35.458665    192.168.11.9     192.168.1
  378 12:40:35.459947    192.168.11.9     192.168.1
  379 12:40:35.461238    192.168.11.9     192.168.1
  380 12:40:35.679395    192.168.11.9     192.168.1
```

```
⊞ Frame 1 (45 bytes on wire, 45 bytes captured)
⊞ Ethernet II, Src: Azurewav_e1:ad:38 (00:15:af:e1:ad:38), Dst: I
⊞ Internet Protocol, Src: 192.168.11.7 (192.168.11.7), Dst: 192.1
⊞ User Datagram Protocol, Src Port: 26039 (26039), Dst Port: 3641
⊞ Data (3 bytes)
```

```
0000  00 21 5d 57 a1 3c 00 15  af e1 ad 38 08 00 45 00   .!]W.<.. ...8..E.
0010  00 1f 64 27 00 00 80 11  3f 46 c0 a8 0b 07 c0 a8   ..d'.... ?F......
0020  0b 09 65 b7 8e 3a 00 0b  8d 72 ba 12 2d            ..e..:.. .r..-
```

Microsoft: <live capture in progress> File: C:... Packets: 380 Displayed: 380 Marked: 0

D:\UniVeRsiTy\Hatfield\Work\apps v.0.2\JRTPTestEvents\Debug\JRTPTest.exe

```
 ----------------------------------------------------
 !                RTP CLIENT - JRtpLib              !
 ----------------------------------------------------

 ----------------------------------------
 Default server port: 4000

 Press 's' to change default settings.

 Press any key to start client.

 ----------------------------------------

>>>> WSAStartup() is OK.

>>>> RTP session created with portbase 4000
>>>> Communicating with server. Please wait...
```

Fig. 6: Capturing packets with WireShark while client application is executing

4.2.2 Qualitative test on video streaming

For a first qualitative glance on the RTP performances some video streaming tests have been done using the developed RTP and TCP streaming applications. In this case, the tests results are not numerical, but looking at the client's side video window, it is possible to evaluate the fluidity of the streaming and the possible delay between the images shown at the server and at the client side.

4.3 Tests results

4.3.1 TCP Synchronous transmission

- Request/Reply – LAN Network – Simulation Time: 30s – Payload size: 128 bytes

Server Sent Bytes	Client Rcv Bytes	Throughput [KB/s]
2303104	2303104	74,971
2863104	2863104	93,2
2688384	2688384	87,513
2747136	2747136	89,425
2736256	2736256	89,071
2608000	2608000	84,896
2406528	2406528	78,338
2739200	2739200	89,167
2669952	2669952	86,912
2653392	2653392	85,787
2692352	2692352	87,642
2880256	2880256	93,758
2828672	2828672	92,079
2830976	2830976	92,154
2899200	2899200	94,375

- Request/Reply – INTERNET Network – Simulation. Time: 30s – Payload size: 128 bytes

Server Sent Bytes	Client Rcv Bytes	Throughput [KB/s]
54144	54144	1,736
57728	57728	1,879
60800	60800	1,979
63488	63488	2,067
68096	68096	2,217
64768	64768	2,108
61696	61696	2,008
58496	58496	1,904
57728	57728	1,879
59264	59264	1,929
51072	51072	1,663
39552	39552	1,288
50304	50304	1,638
55552	55552	1,808
69248	69248	2,254

4.3.2 TCP Asynchronous transmission

❖ LAN Network – Simulation Time: 30s – Payload size: 128 bytes

Server Sent Bytes	Client Rcv Bytes	Throughput [KB/s]
40383744	40349696	1313,467
40173952	40136884	1306,539
40417664	40381056	1314,488
49665920	49631920	1615,622
49538048	49498168	1611,268
39885312	39847456	1297,118
39967616	39930656	1299,826
50096256	50057892	1629,489
35121152	35083776	1142,05
40495488	40458368	1317,004
40710016	40674048	1324,025
39892480	39857536	1297,446
50222976	50187476	1633,707
39405824	39370240	1281,583
50180480	50140644	1632,182

❖ INTERNET Network – Simulation Time: 30s – Payload size: 128 bytes

Server Sent Bytes	Client Rcv Bytes	Throughput [KB/s]
851584	843520	27,458
818816	810752	26,392
852736	844544	27,492
875520	867456	28,238
864128	856064	27,867
862592	854528	27,817
856960	848896	27,633
845312	835840	27,208
854528	846336	27,55
869248	861184	28,033
868096	860032	27,996
854272	844800	27,5
861312	853120	27,771
876416	868352	28,267
861824	852224	27,742

4.3.3 RTP Transmission

❖ LAN Network – Simulation Time: 30s – Payload size: 128 bytes

Server Sent Bytes	Client Rcv Bytes	Sent Packets	Rcv Packets	Throughput [KB/s]	Interrarival Jitter [s]
8854144	8568448	69173	66941	278,921	0,000921875001746
9116800	8542848	71225	66741	278,087	0,000921875001746
9047936	8782336	70687	68612	285,883	0,000859374998108
9047936	8792960	70687	68695	286,229	0,000921874999927
9000576	8764416	70317	68472	285,3	0,000859375000155
8682624	8428160	67833	65845	274,354	0,000859374998108
9099904	8568320	71093	66940	278,917	0,000859375000837
8881408	8616832	69386	67319	280,496	0,000859374999927
9049984	8792064	70703	68688	286,2	0,000921874999927
8872704	8616704	69318	67318	280,492	0,000921874999927
9158784	8569472	71553	66949	278,954	0,000859374998108
8649984	8398336	67578	65612	273,383	0,000859375001746
8963456	8430720	70027	65865	274,438	0,000921875001746
9182720	8922624	71740	69708	290,45	0,000859374999927
7731456	7477504	60402	58418	243,408	0,000859374998108

❖ INTERNET Network – Simulation Time: 30s – Payload size: 128 bytes

Server Sent Bytes	Client Rcv Bytes	Sent Packs	Rcv Packs	Throughput [KB/s]	Interrarival Jitter [s]
11815168	1527936	92306	11937	49,737	0,000203125000364
11978752	1560064	93584	12188	50,783	0,000531249999563
14016384	1608192	109503	12564	52,35	0,000859374998108
14037760	1608704	109670	12568	52,367	0,000140625001019
13961344	1596160	109073	12470	51,958	0,000015624999127
13644416	1619456	106597	12652	52,717	0,000000000000946
13922688	1605888	108771	12546	52,275	0,000218750001091
13997184	1605504	109353	12543	52,263	0,000234375001164
13791744	1599616	107748	12497	52,071	0,000859374998108
14266752	1594368	111459	12456	51,9	0,000296874997527
13804544	1616896	107848	12632	52,633	0,000375000003056
13902208	1592320	108611	12440	51,833	0,000234374999345
13995520	1625472	109340	12699	52,913	0,000140625001019
13048192	1552256	101939	12127	50,529	0,000000000000077
14519808	1624192	113436	12689	52,871	0,000078125001019

4.4 Statistics

4.4.1 TCP Synchronous transmission

- ❖ Request/Reply – LAN network – Simulation Time: 30s – Payload size: 128 bytes

	Server Sent Bytes	**Client Rcv Bytes**	**Throughput [KB/s]**
Average	2703100,8	2703100,8	87,95253333
Standard Deviation	167502,7831	167502,7831	5,466947776

- ❖ Request/Reply – INTERNET Network – Simulation Time: 30s – Payload size: 128 bytes

	Server Sent Bytes	**Client Rcv Bytes**	**Throughput [KB/s]**
Average	58129,06667	58129,06667	1,890466667
Standard Deviation	7540,503275	7540,503275	0,24639996274

The WireShark capture files confirm the results obtained through the developed applications, in terms of number of bytes exchanged between client and server and average throughput.

Figure 7 shows one of the statistics summary obtained through WireShark for the TCP synchronous traffic simulation. Notice that this tool shows the results considering the whole frame size, not just the payload as in the described applications. That's why some differences between values can be noticed.

4.4.2 TCP Asynchronous transmission

❖ LAN Network – Simulation Time: 30s – Payload size: 128 bytes

	Server Sent Bytes	**Client Rcv Bytes**	**Throughput [KB/s]**
Average	43077128,53	43040387,73	1401,054267
Standard Deviation	5194385,183	5193905,32	169,0723411

❖ INTERNET Network – Simulation Time: 30s – Payload size: 128 bytes

	Server Sent Bytes	**Client Rcv Bytes**	**Throughput [KB/s]**
Average	858222,9333	849843,2	27,66426667
Standard Deviation	14016,09324	14124,87809	0,459845699

The statistics summary obtained through WireShark for the TCP asynchronous traffic simulation on LAN are shown in Figure 8.

	Server Sent Bytes	**Client Rcv Bytes**	**Sent Packets**	**Rcv Packets**	**Throughput[KB/s]**	**Interarrival Jitter [s]**
Average	8889361,067	8551449,6	69448,13333	66808,2	278,3674667	0,000884375000003
Standard Deviation	357673,2578	334743,0196	2794,322327	2615,179841	10,89664549	0,000031693285257

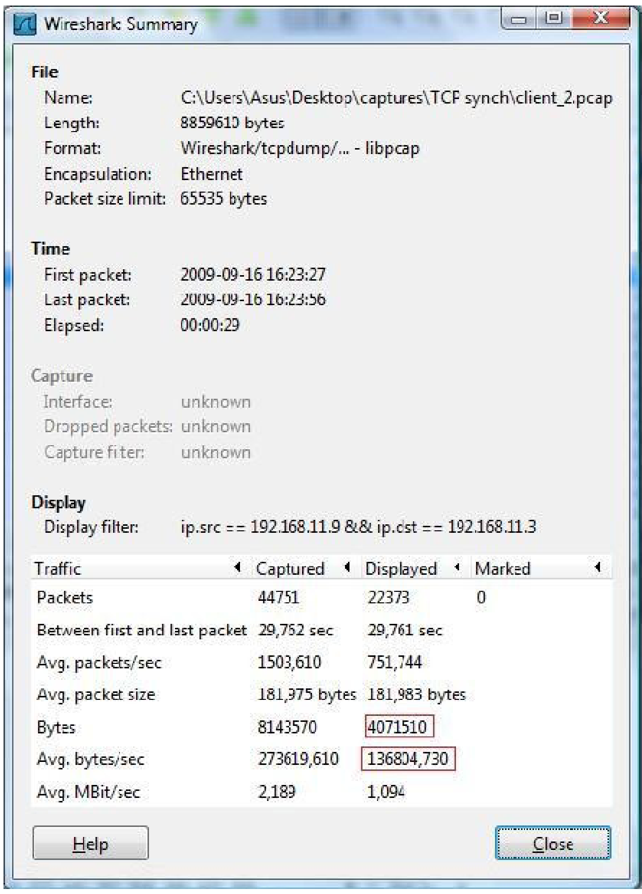
4.4.3 RTP transmission

❖ LAN network – Simulation Time: 30s – Payload size: 128 bytes

❖ INTERNET Network – Simulation Time: 30s – Payload size: 128 bytes

	Server Sent Bytes	**Client Rcv Bytes**	**Sent Packs**	**Rcv Packs**	**Throughput [KB/s]**	**Interrarival Jitter [s]**
Average:	13646830,93	1595801,6	106615,8667	12467,2	51,9467	0,000279166666769
Standard Deviation	777453,467	28049,3285	6073,855211	219,1353789	0,913278917392104	0,000275668

The statistics summary obtained through WireShark for the RTP traffic simulation are shown in Figure 9.



Wireshark Summary

File

Name: C:\Users\Asus\Desktop\captures\TCP synch\client_2.pcap
Length: 8859610 bytes
Format: Wireshark/tcpdump/... - libpcap
Encapsulation: Ethernet
Packet size limit: 65535 bytes

Time

First packet: 2009-09-16 16:23:27
Last packet: 2009-09-16 16:23:56
Elapsed: 00:00:29

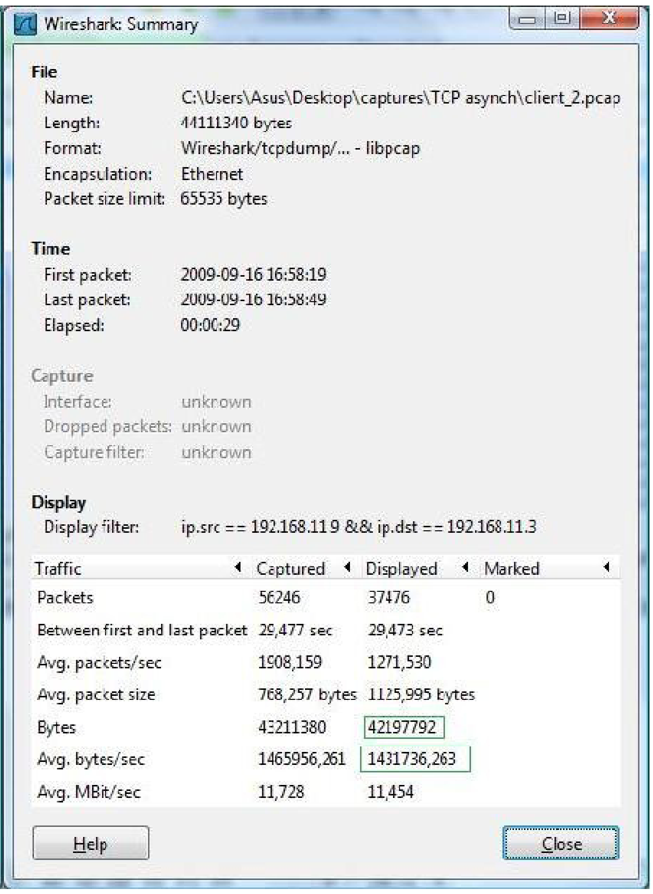
Capture

Interface: unknown
Dropped packets: unknown
Capture filter: unknown

Display

Display filter: ip.src == 192.168.11.9 && ip.dst == 192.168.11.3

Traffic	Captured	Displayed	Marked
Packets	44751	22373	0
Between first and last packet	29,752 sec	29,761 sec	
Avg. packets/sec	1503,610	751,744	
Avg. packet size	181,975 bytes	181,983 bytes	
Bytes	8143570	4071510	
Avg. bytes/sec	273619,610	136804,730	
Avg. MBit/sec	2,189	1,094	



Wireshark Summary

File

Name: C:\Users\Asus\Desktop\captures\TCP asynch\client_2.pcap
Length: 44111340 bytes
Format: Wireshark/tcpdump/... - libpcap
Encapsulation: Ethernet
Packet size limit: 65535 bytes

Time

First packet: 2009-09-16 16:58:19
Last packet: 2009-09-16 16:58:49
Elapsed: 00:00:29

Capture

Interface: unknown
Dropped packets: unknown
Capture filter: unknown

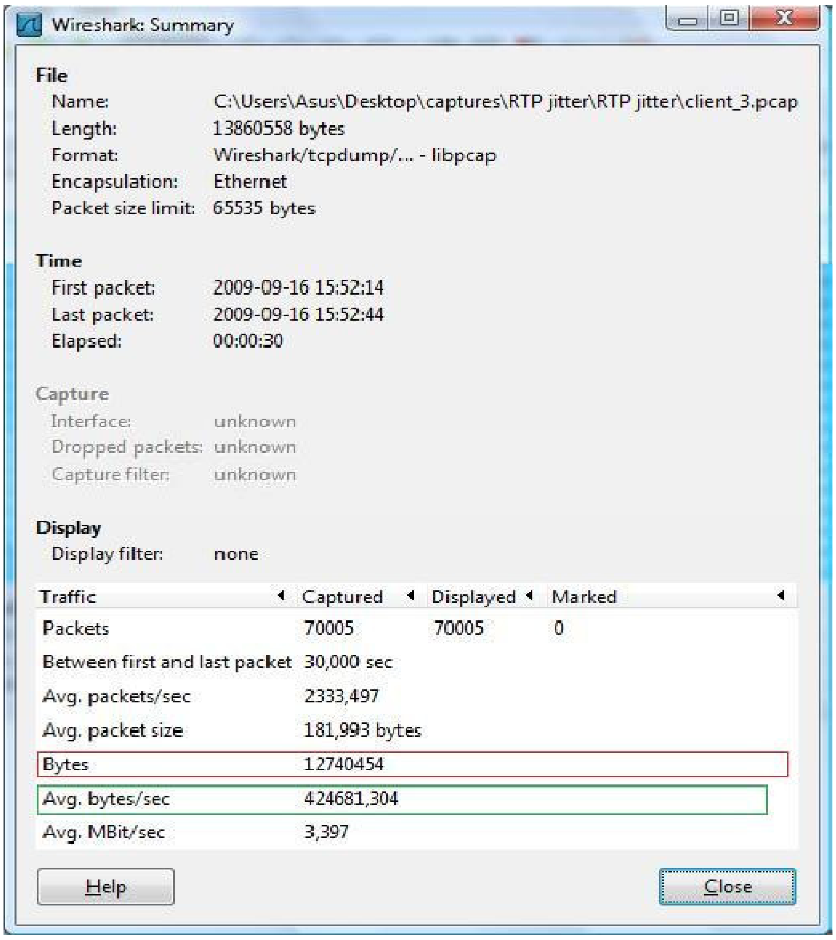
Display

Display filter: ip.src == 192.168.11.9 && ip.dst == 192.168.11.3

Traffic	Captured	Displayed	Marked
Packets	56246	37476	0
Between first and last packet	29,477 sec	29,473 sec	
Avg. packets/sec	1908,159	1271,530	
Avg. packet size	768,257 bytes	1125,995 bytes	
Bytes	43211380	42197792	
Avg. bytes/sec	1465956,261	1431736,263	
Avg. MBit/sec	11,728	11,454	

Figure 7: Synchronous TCP Statistics Summary on LAN

Figure 8: Asynchronous TCP Statistics Summary on LAN



Wireshark: Summary

File

Name:	C:\Users\Asus\Desktop\captures\RTP jitter\RTP jitter\client_3.pcap
Length:	13860558 bytes
Format:	Wireshark/tcpdump/... - libpcap
Encapsulation:	Ethernet
Packet size limit:	65535 bytes

Time

First packet:	2009-09-16 15:52:14
Last packet:	2009-09-16 15:52:44
Elapsed:	00:00:30

Capture

Interface:	unknown
Dropped packets:	unknown
Capture filter:	unknown

Display

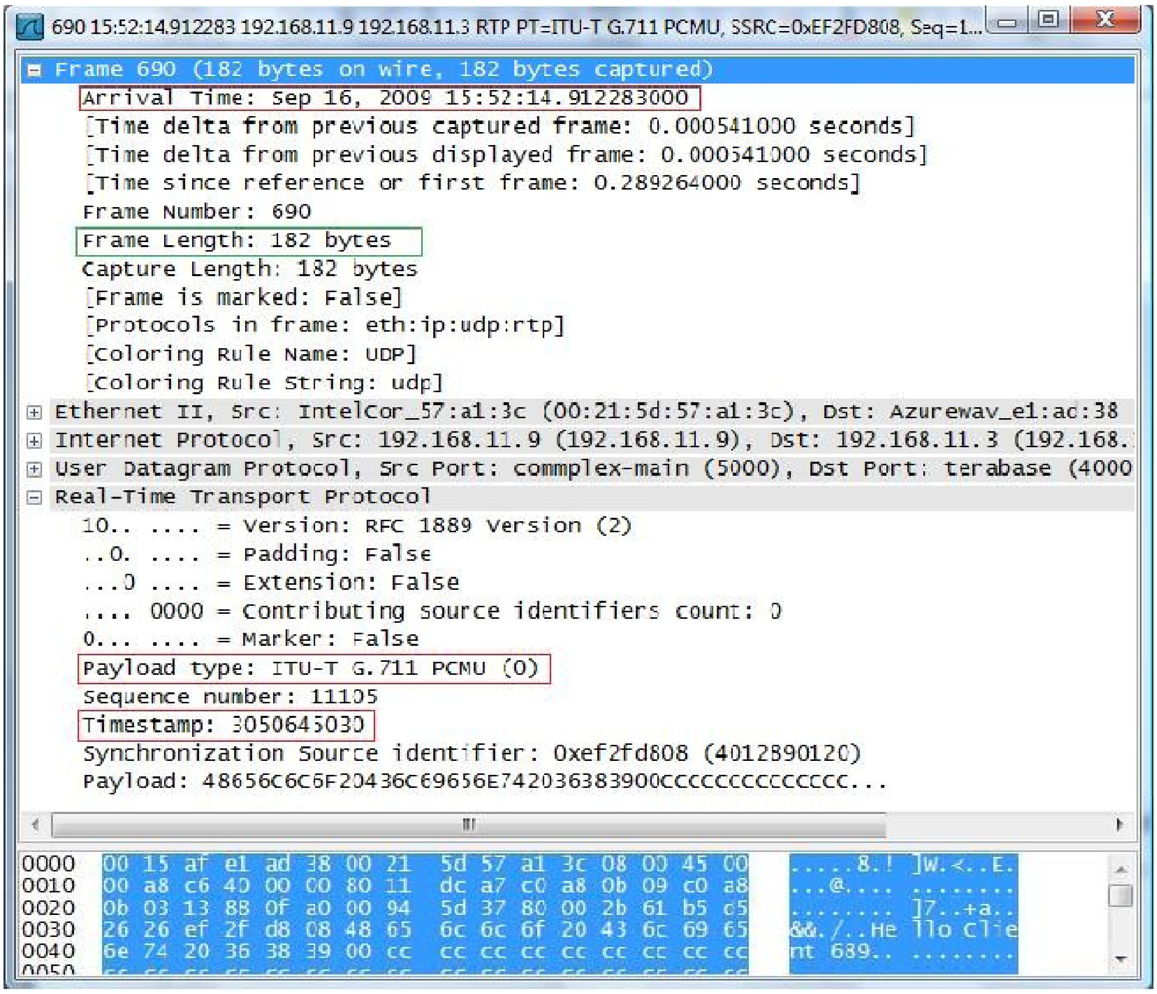
Display filter:	none

Traffic	Captured	Displayed	Marked
Packets	70005	70005	0
Between first and last packet	30,000 sec		
Avg. packets/sec	2333,497		
Avg. packet size	181,993 bytes		
Bytes	12740454		
Avg. bytes/sec	424681,304		
Avg. MBit/sec	3,397		

Figure 9: WireShark RTP traffic Statistics Summary on LAN

With WireShark it is also possible to decode packets on the basis of their protocol, in this case we used RTP protocol for the RTP application tests. In fact, WireShark can capture network traffic filtering with a given protocol (i.e. UDP). After that using the "*decode as...*" menu item it's possible to decode the captured frames for the specific protocol, i.e. RTP.

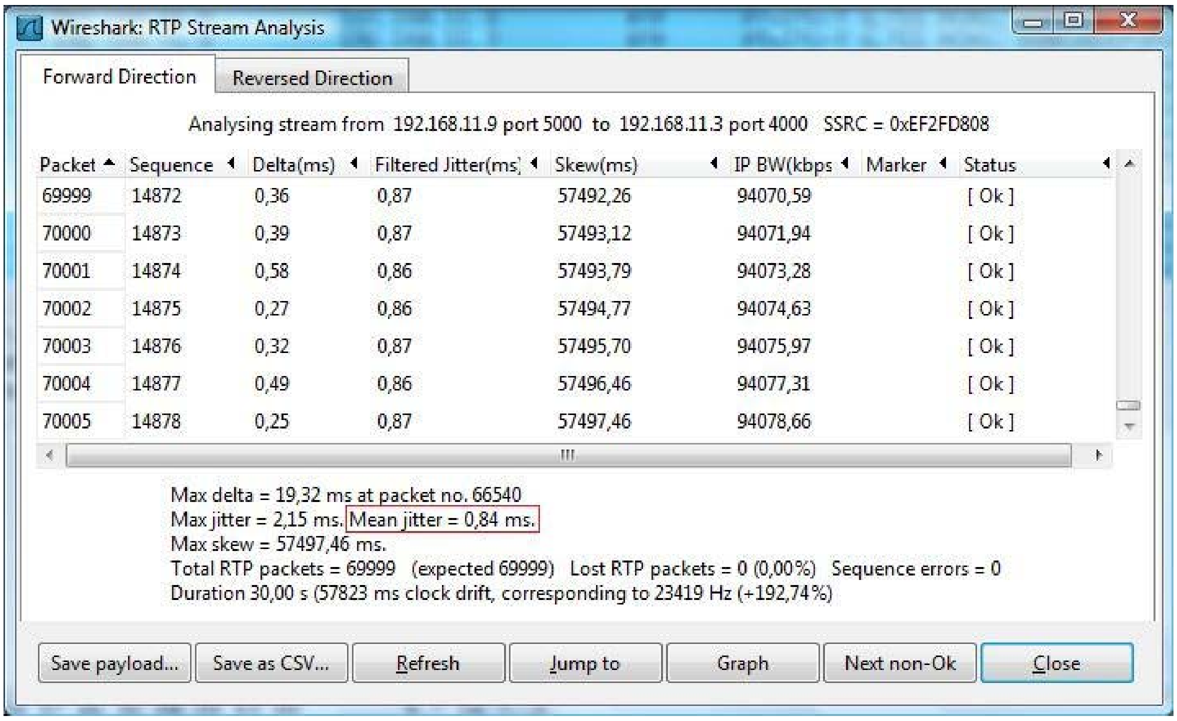
This allows the user to obtain a detailed packet description window, like the one in Figure 10. The WireShark frame details window shows a number of useful information which can be used to provide some statistics: for example the items in the red box can be used to evaluate the interarrival jitter. Besides, once all frames have been decoded as RTP traffic, we can analyze the captured RTP stream, through the window shown in Figure 11.



```
690 15:52:14.912283 192.168.11.9 192.168.11.3 RTP PT=ITU-T G.711 PCMU, SSRC=0xEF2FD808, Seq=1...
⊟ Frame 690 (182 bytes on wire, 182 bytes captured)
    Arrival Time: Sep 16, 2009 15:52:14.912283000
    [Time delta from previous captured frame: 0.000541000 seconds]
    [Time delta from previous displayed frame: 0.000541000 seconds]
    [Time since reference or first frame: 0.289264000 seconds]
    Frame Number: 690
    Frame Length: 182 bytes
    Capture Length: 182 bytes
    [Frame is marked: False]
    [Protocols in frame: eth:ip:udp:rtp]
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]
⊞ Ethernet II, Src: IntelCor_57:a1:3c (00:21:5d:57:a1:3c), Dst: Azurewav_e1:ac:38
⊞ Internet Protocol, Src: 192.168.11.9 (192.168.11.9), Dst: 192.168.11.3 (192.168.
⊞ User Datagram Protocol, Src Port: commplex-main (5000), Dst Port: terabase (4000
⊟ Real-Time Transport Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 .... = Extension: False
    .... 0000 = Contributing source identifiers count: 0
    0... .... = Marker: False
    Payload type: ITU-T G.711 PCMU (0)
    Sequence number: 11105
    Timestamp: 3050645030
    Synchronization Source identifier: 0xef2fd808 (4012890120)
    Payload: 48656C6C6F20436C69656E74203638390CCCCCCCCCCCCCC...

0000  00 15 af e1 ad 38 00 21  5d 57 a1 3c 08 00 45 00   .....8.! ]W.<..E.
0010  00 a8 c6 40 00 C0 80 11  dc a7 c0 a8 0b 09 c0 a8   ...@.... ........
0020  0b 03 13 88 0f a0 00 94  5d 37 80 00 2b 61 b5 c5   ........ ]7..+a..
0030  26 26 ef 2f d8 08 48 65  6c 6c 6f 20 43 6c 69 65   &&./..He llo Clie
0040  6e 74 20 36 38 39 00 cc  cc cc cc cc cc cc cc cc   nt 689.. ........
```

Figure 10: WireShark RTP frame details on LAN



Wireshark: RTP Stream Analysis

Forward Direction | Reversed Direction

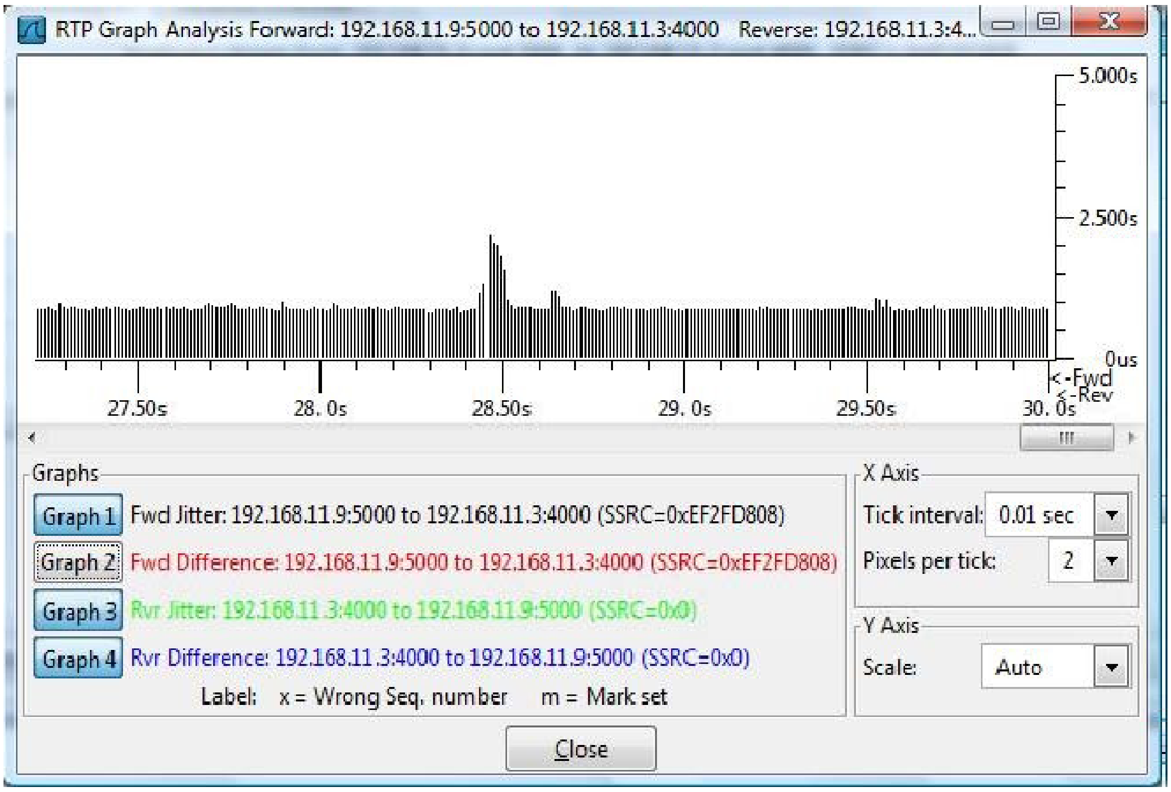
Analysing stream from 192.168.11.9 port 5000 to 192.168.11.3 port 4000 SSRC = 0xEF2FD808

Packet	Sequence	Delta(ms)	Filtered Jitter(ms)	Skew(ms)	IP BW(kbps)	Marker	Status
69999	14872	0,36	0,87	57492,26	94070,59		[Ok]
70000	14873	0,39	0,87	57493,12	94071,94		[Ok]
70001	14874	0,58	0,86	57493,79	94073,28		[Ok]
70002	14875	0,27	0,86	57494,77	94074,63		[Ok]
70003	14876	0,32	0,87	57495,70	94075,97		[Ok]
70004	14877	0,49	0,86	57496,46	94077,31		[Ok]
70005	14878	0,25	0,87	57497,46	94078,66		[Ok]

```
Max delta = 19,32 ms at packet no. 66540
Max jitter = 2,15 ms. Mean jitter = 0,84 ms.
Max skew = 57497,46 ms.
Total RTP packets = 69999   (expected 69999)   Lost RTP packets = 0 (0,00%)   Sequence errors = 0
Duration 30,00 s (57823 ms clock drift, corresponding to 23419 Hz (+192,74%)
```

Figure 11: WireShark RTP stream analisys on LAN

Notice that the WireShark computed interarrival jitter is about the same as the one obtained through the developed applications. The stream analysis also shows the source and destination addresses and port, the SSRC for each stream and the eventually lost packets or sequence errors detected during the simulation. The jitter information can be plotted in a proper graph, as shown in Figure 12. The graph shows the values of the interarrival jitter (computed for each incoming frame).



Fwd Jitter: 192.168.11.9:5000 to 192.168.11.3:4000 (SSRC=0xEF2FD808)
Fwd Difference: 192.168.11.9:5000 to 192.168.11.3:4000 (SSRC=0xEF2FD808)
Rvr Jitter: 192.168.11.3:4000 to 192.168.11.9:5000 (SSRC=0x0)
Rvr Difference: 192.168.11.3:4000 to 192.168.11.9:5000 (SSRC=0x0)
Label: x = Wrong Seq. number m = Mark set

Tick interval: 0.01 sec
Pixels per tick: 2
Scale: Auto

Figure 12: Interarrival jitter graph

Chapter 5
Conclusion and Future Works

The main aim of this work was investigating on the RTP real-time transport protocol in order to improve the performances of the MO.R.D.U.C. client- server communication, which actually use HTTP requests over TCP protocol for exchanging data and commands. The first step has been performing various tests on both the RTP and TCP client-server applications for estimating the average throughput, measured in KBytes per second. Tests have been performed both on LAN network and on the Internet.

Even if the average throughput obtained with the RTP protocol over LAN connection is less than the one obtained using TCP, tests performed on the internet demonstrate that the real-time transport protocol provides better performances: throughput results almost twice the one provided by TCP.

Besides, qualitative tests performed with video streaming applications, have confirmed this result, providing better streaming fluidity and a lower delay.

Finally, the achieved results have confirmed the idea that the RTP protocol is suitable for real-time applications, such as the MO.R.D.U.C. system, and allows a better timing on communication.

Future work will focus on the development and design of an appropriate communication protocol for commands and for the camera, laser and odometric data exchange between MO.R.D.U.C. server and client. The main aim is to obtain an efficient protocol which can represent data in a short way, giving information about timestamps and allowing the synchronism for the data exchange itself.

Besides, the majority of RTP implementations use the RTSP protocol (a multimedia streaming application-level protocol) for request-based data exchange.

Since the current MO.R.D.U.C. system is based on HTTP requests and the RTSP is similar in syntax and operations to it, providing RTP/RTSP client-server communication will be the best solution. In fact, RTP sessions and RTSP requests make these protocols to be suitable for MO.R.D.U.C. application, adopting two distinct sessions for commands and sensors data exchange.

That's why the final target will be to modify properly the existent MO.R.D.U.C. server (and the client too), using these real-time protocols, making it to be more efficient and providing better performances for a fluid and synchronized vision.

Bibliography

[1] Microrobot teleoperation through WWW András Lassó, Tamás Urbancsek, Ádám Helybély Budapest University of Technology and Economics. Department of Control Engineering and Information Technology.
[2] Encyclopaedia Britannica, http://www.britannica.com/, 2007.
[3] Internet – Based Telerobotics: Problems and Approches, Paolo Fiotini, Roberto Oboe Università di Padova.
[4]: Developing and experimenting on-line mobile robot teleguide. D. Di Mauro
[5]: Telerobotics: Through−The−Internet Teleoperation of the ABB IRB 2000 Industrial Robot. Alberto J. Álvares, Guilherme C. de Carvalho, Luis Felipe A. Paulinyi & Sadek C. A. Alfaro. Automation and Control Group, Mechanical Engineering Department, University of Brasilia
[6] : Il protocollo di trasporto Rtp http://www.faqs.org/rfcs/rfc3550.html
[7]: Delivery of Real-time Continuous Media over the Internet. Randa El-Marakby, David Hutchison
[8] RTSP Interoperability with real system server 8
[9]: Sick Germany 2D-Laser-scanner LMS200/211/221/291 Data Sheet
[10]: S. Livatino and G. Muscato (supervisors), F. Privitera (student), "3D Stereo Visualization advantages for the mobile robot teleguide" Report Medialogy Semester 9,December 2004.
[11]: Mobile Robot tele-guide based on laser sensors. V. Neri
[12] Jrtplib online documentation, http://research.edm.uhasselt.be
[13] RTP performances metrics, http://www.ietf.org
[14] RTP protocol interarrival jitter – RFC 1889, http://www.faqs.org/rfcs
[15] Tomi Yletyinen and Raimo Kantola, *Voice packet interarrival jitter over IP switching* Helsinki University of Technology, Laboratory of Telecommunications Technology
[16] *WireShark network protocol analyzer*, http://www.wireshark.org