Improving Telecommunication Performance for Robot Teleguide: a study on the use of TCP and RTP Protocols

Simona Ullo

University of Hertfordshire
School of Electronic Communication and Electrical Engineering
Hatfield, UK

Supervisor: Prof. S.Livatino

September, 2009

September 21, 2009

Contents

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. It has been deve loped with the support of Prof. S.Livatino who's currently working at the School of Electronic, Communication and Electrical Engineering at the University of Hertfordshire, United Kingdom.
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, implementation and test of a real-time communication protocol applied to telerobotics systems. In fact, in order to improve MO.R.D.U.C 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 multimedia streaming protocol, the project focuses are:

1. Study of RTP protocol for real-time applications;

2. Development of RTP and TCP-based client-server applications for data

exchange and video streaming;

3. Performing tests on developed applications for evaluating protocols performances, providing numerical statistics for the comparison between TCP and RTP.

Chapter 1

Applications description

1.1 RTP event driven client-server application

This client-server application exchanges simple string data through the RTP protocol. It has been developed using the Jrtplib C++ framework [2] which implements all the RTP protocol [3] features described in RFC 3550 [4].
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 destination 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 developed 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.
At the client side, after the session establishment nothing is done until the

packets are received. The client behaviour is event driven, i.e. a defined event handler executes whenever a new RTP packet reception event occurs. The handler keeps count of the total number of received bytes and packets, manages the elapsed time from the start of simulation and performs the required calculation to provide the interarrival jitter.

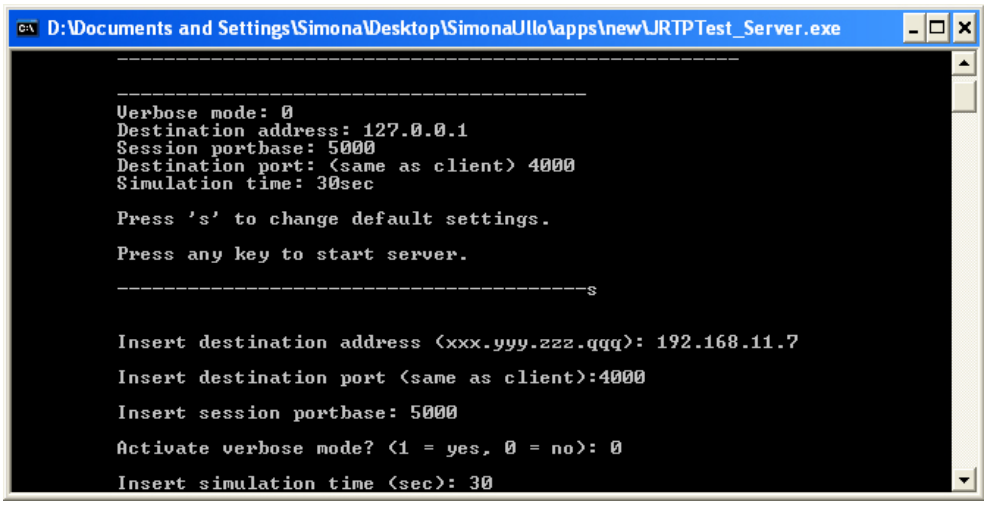

Figure 1.1: RTP server configuration

The simulation timer starts at the reception of the first RTP packet. At the end of simulation the client shows the statistical information about the number of packets and bytes received from the server, the average throughput and the interarrival jitter [5][6][7]. 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.

1.2 TCP synchronous client-server application

This client-server application has been developed using Microsoft Winsock2 socket management so that the client can connect to the server just knowing the pair IP address and port. The developed applications just follow the standard socket programming methodology, allowing the user to set the following parameters:

- Server connection IP and port (cliend 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.

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 conseguence 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.

1.4 RTP video streaming client-server application

This client-server application provides a video streaming using the RTP protocol. It has been developed using the Jrtplib and the OpenCV [9] frameworks (for camera capture). Video frames are captured at server side via a web camera and are sent to the client in real-time. The RTP client and server are defined in a similar way than the RTP simple data exchange application, but in this case the server has to 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). This is required because the RTP protocol uses the UDP transport protocol which has a frame size limit of 65535 bytes. Since the OpenCV framework provides video frames in the IplImage format, which couldn't be sent on the network without encoding it properly, the server saves each captured frame in the jpeg format, before sending it to the client.
This approach causes an additional overhead due to the disk file access (to store and load the jpeg frame) and can be avoided using any jpeg encoding algorithm, directly from the IplImage data type. Anyway, this doesn't compromise the comparison statistics, because both the TCP and the RTP streaming applications use the same approach. At the client side, there is the previously described event handler which has to build the jpeg frame from the received bytes and show it through an OpenCV provided window.

1.5 TCP video streaming client-server application

This client-server application provides a video streaming using the TCP protocol. It has been developed using the Microsoft Winsock2 sockets and the OpenCV framework. Since the application has to perform a video streaming, the TCP protocol is used in asynchronous mode. The capture and encoding/decoding of video frames is provided in the same way as the previous RTP streaming application.
The main difference consists in the way the client receives the captured data. In fact, due to the TCP winsock programming, a packet (video frame) can be received with the recv primitive call, by which a given buffer is filled with data, if any. The buffer size must be defined in a static way but we don't know how long the payload is, before receiving it.
So we have to choose between two possibilities: set the buffer size to a small number of Kbytes, repeating the recv call until the entire frame has been received (requires a flag which indicates the end of a video frame) or oversize the buffer itself, so that a single frame can be received at a time. The choice between the two approaches has been made on the basis of the following consideration.
The TCP video streaming application needs to store each video frame on disk in a jpeg format and then reload it for the transmission over the network. This causes a delay due to the disk file access which produces low quality video streaming. The smallest is the buffer size (first approach) for the TCP recv function call, the worst is the video quality, because more disk accesses

are required. In order to compare TCP and RTP under the same conditions this overhead must be reduced to the minimum (i.e. one disk access per frame). That's the reason why the former approach has been used and the buffer has been oversized up to 20KB, to surely contain the biggest video frame.

Chapter 2

Test methodology

2.1 Throughput and interarrival jitter evaluation

In order to obtain a quantitative comparison between TCP and RTP protocols, the developed 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. In addition to the statistics provided by the applications, a sniffing software called WireShark [8] 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.
The tests have been done on the following three cases:

- RTP traffic
- TCP asynchronous traffic
- TCP synchronous traffic

Each of them has been tested performing 15 applications runs, both using LAN and internet connection, and providing the following statistical information:

- For TCP applications:
  - Number of bytes sent by the server
  - Number of bytes received from the client

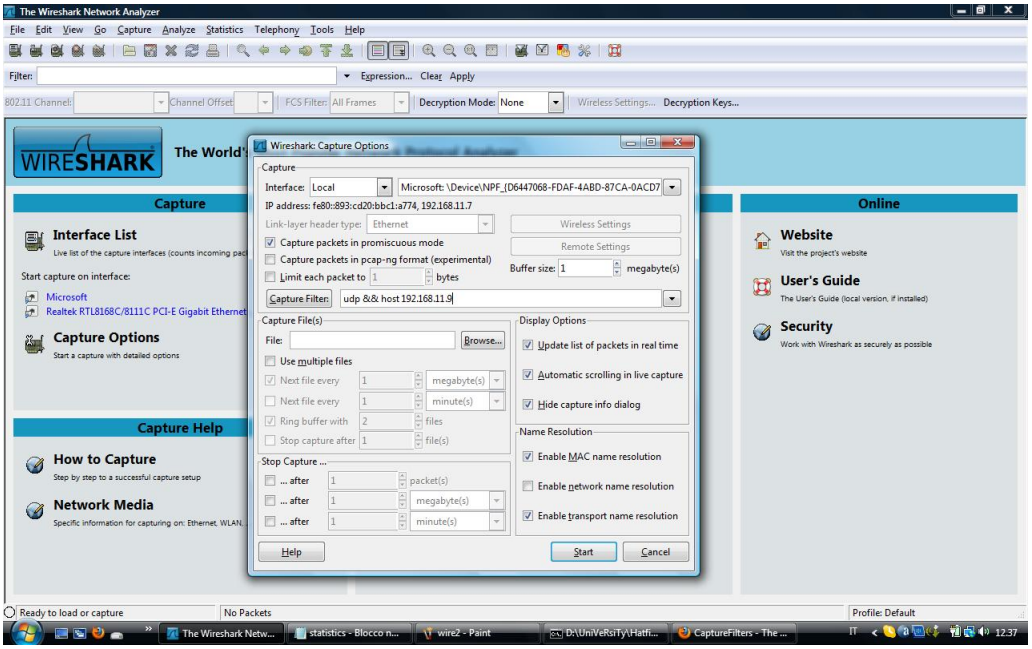

Figure 2.1: WireShark capturing filter for RTP packets only

  - 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]

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

2.2 Qualitative test on video streaming

For giving a first qualitative glance at 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 fuidity of the streaming and the possible delay between the images shown at the server and at the client side.

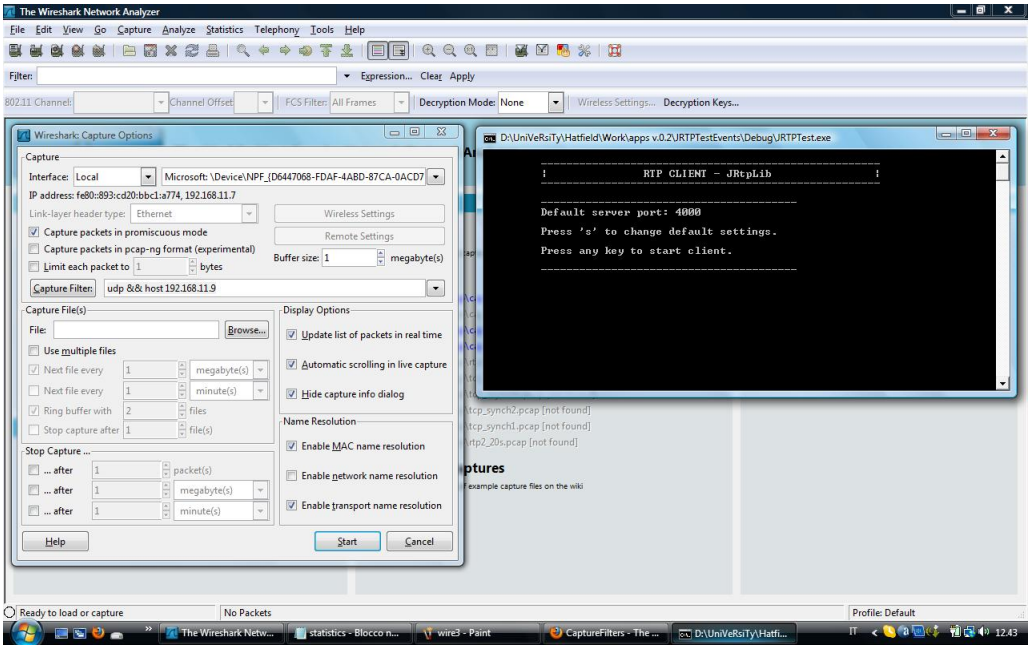

Figure 2.2: Running client application and starting capture with WireShark

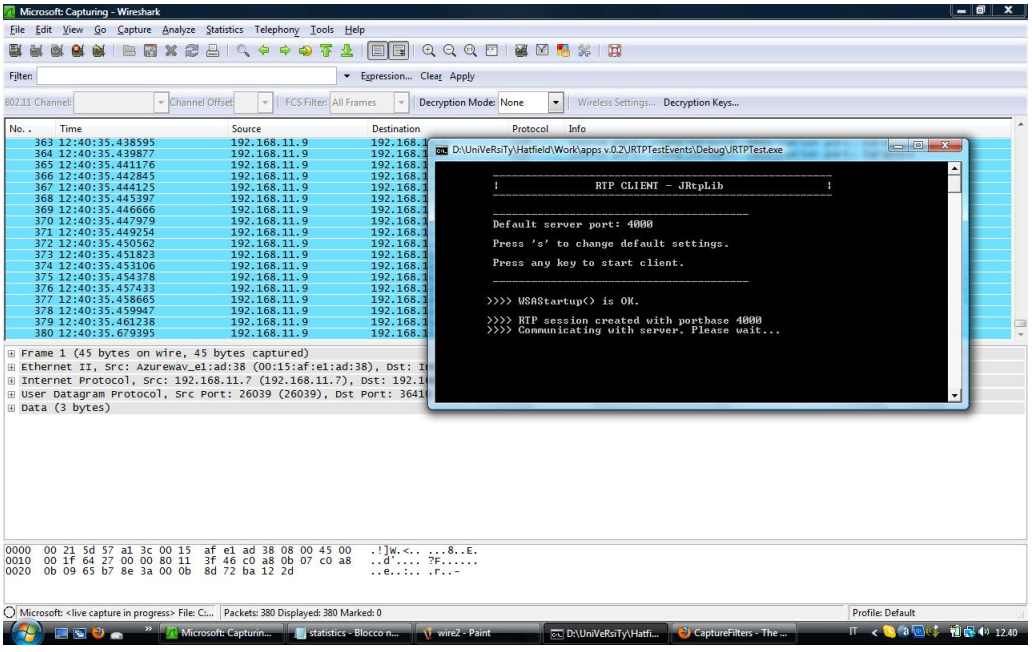

Figure 2.3: Capturing packets with WireShark while client application is executing

Chapter 3

Tests results

3.1 TCP Synchronous trasmission

Case 1: Request/Reply LAN network Sim. Time: 30s Payload size: 128 bytes
This first test shows the TCP protocol performances achieved running a client-server application with a request-reply synchronous approach, over LAN network. The obtained values are shown in figure 3.1.

Case 2: Internet connection Sim. Time: 30s Payload size: 128 bytes
Figure 3.2 shows the transmission results over internet connection.

3.2 TCP Asynchronous trasmission

Case 1: LAN network Sim. Time: 30s Payload size: 128 bytes
This test refers to a client-server TCP-based application using an asynchronous communication approach, which allows the server of continuously sending data to the client, after establishing the connection. Results over LAN network are shown in figure 3.3.

Case 2: Internet connection Sim. Time: 30s Payload size: 128 bytes
Figure 3.4 shows the results for TCP asynchronous communication obtained through internet connection.

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

Figure 3.1: TCP synchronous transmission results over LAN

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

Figure 3.2: TCP synchronous transmission results over internet

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

Figure 3.3: TCP asynchronous transmission results over LAN

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

Figure 3.4: TCP asynchronous transmission results over internet

Server Sent Bytes	Client Rcv Bytes	Sent Packets	Rcv Packets	Throughput [KB/s]	Interarrival 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

Figure 3.5: RTP transmission results over LAN

3.3 RTP trasmission

Case 1: LAN network Sim. Time: 30s Payload size: 128 bytes
This test concern the RTP-based client-server application, run over LAN connection with a 30 seconds simulation time. Results are shown in figure 3.5.

Case 2: Internet connection Sim. Time: 30s Payload size: 128 bytes
Results related to RTP protocol over internet are shown in figure 3.6.

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

Figure 3.6: RTP transmission results over internet

Chapter 4

Statistics

4.1 TCP Synchronous trasmission

Case 1: Request/Reply LAN network Sim. Time: 30s Payload size: 128 bytes
Figure 4.1 shows the statistics obtained testing the TCP protocol using a request-reply approach. This approach allows the server for sending data at the client speed, so the traffic is smoothed down and the achieved throughput is the lowest one compared with the other two cases.

The WireShark capture files confirm the results obtained through the devel-

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

Figure 4.1: TCP synchronous transmission statistics over LAN

oped applications, in terms of number of bytes exchanged between client and server and average throughput. Figure 4.2 shows on the left 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.

Case 2: Internet connection Sim. Time: 30s Payload size: 128 bytes
Testing the synchronous TCP protocol over the internet results in a dramatic reduction of the average throughput, as you can see in figure 4.3.

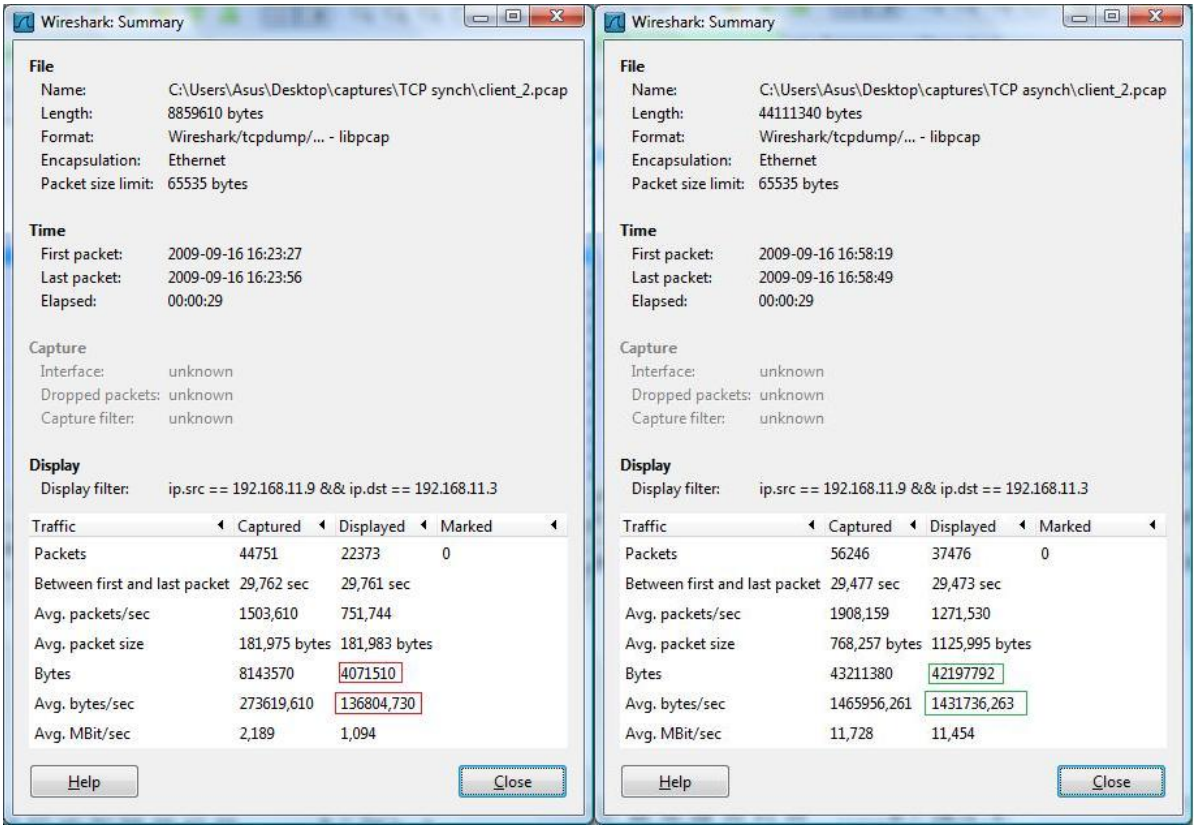

Figure 4.2: TCP WireShark Statistics Summary

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

Figure 4.3: TCP synchronous transmission statistics over internet

4.2 TCP Asynchronous trasmission

Case 1: LAN network Sim. Time: 30s Payload size: 128 bytes The statistics obtained running the TCP asynchronous client-server application are shown in figure 4.4. The asynchronous approach allows to achieve a better average throughput, since the data transmission doesn't require any client request.

The statistics summary obtained through WireShark for the TCP asyn-

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

Figure 4.4: TCP asynchronous transmission statistics over LAN

chronous traffic simulation are shown on the right in Figure 4.2.
Case 2: Internet connection Sim. Time: 30s Payload size: 128 bytes Statistics achieved testing the TCP asynchronous approach over the internet are shown in figure 4.5. Notice that running the client-server applications using internet connection causes a big throughput reduction, due to the transmission latency.

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

Figure 4.5: TCP asynchronous transmission statistics over internet

4.3 RTP trasmission

Case 1: LAN network Sim. Time: 30s Payload size: 128 bytes The RTP-based application tests statistics over LAN are shown in figure 4.6. The

	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

Figure 4.6: RTP transmission statistics over LAN

statistics summary obtained through WireShark for the RTP traffic simulation are shown in Figure 4.7.

Case 2: Internet connection Sim. Time: 30s Payload size: 128 bytes

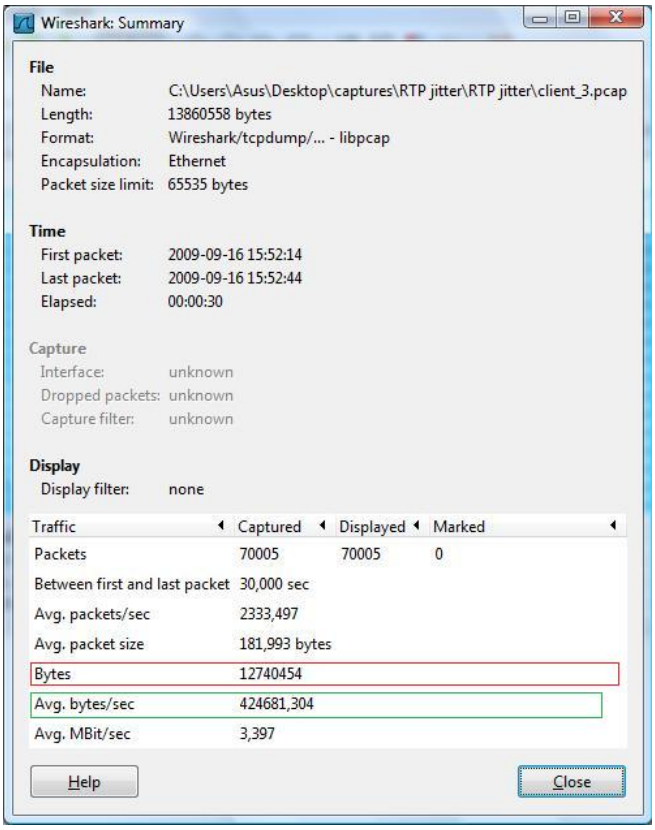

Figure 4.7: WireShark RTP traffic Statistics Summary

The statistics obtained testing RTP protocol over the internet are shown in

figure 4.8. Notice that, using RTP over LAN, it's possible to obtain a lower throughput compared to the one achieved using TCP protocol. Anyway, the more interesting internet statistics highlight a visible superiority (in terms of performances throughput) of the RTP protocol respect to the TCP, as expected. With WireShark it is also possible to decode packets on the basis

	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

Figure 4.8: RTP transmission statistics over internet

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 4.9.
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 4.10.

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 4.11. The graph shows the values of the interarrival jitter (computed for each incoming frame).

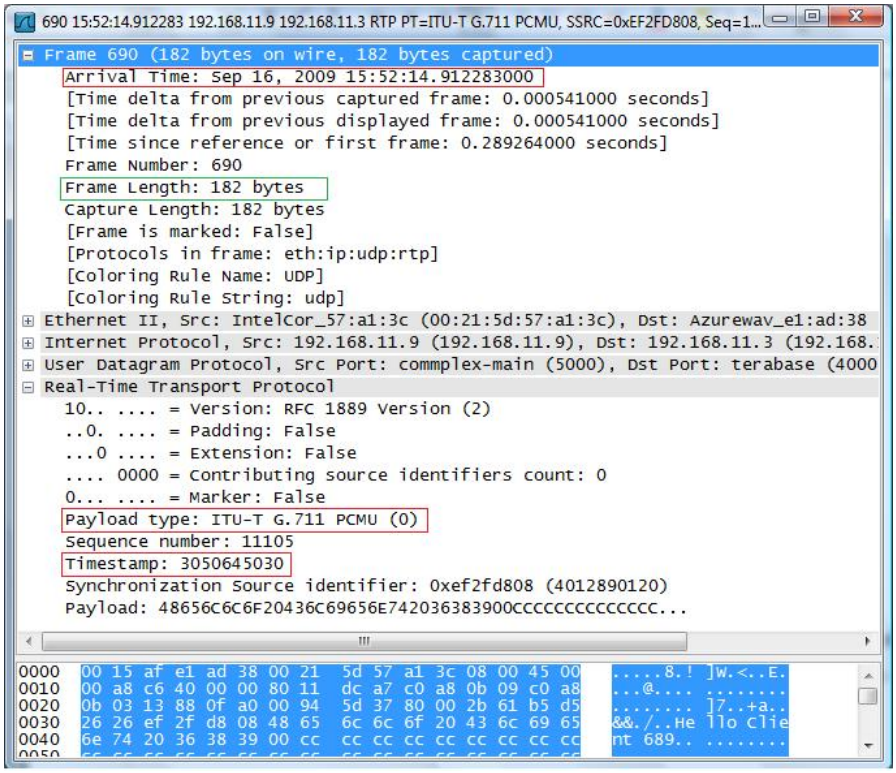

Figure 4.9: WireShark RTP frame details

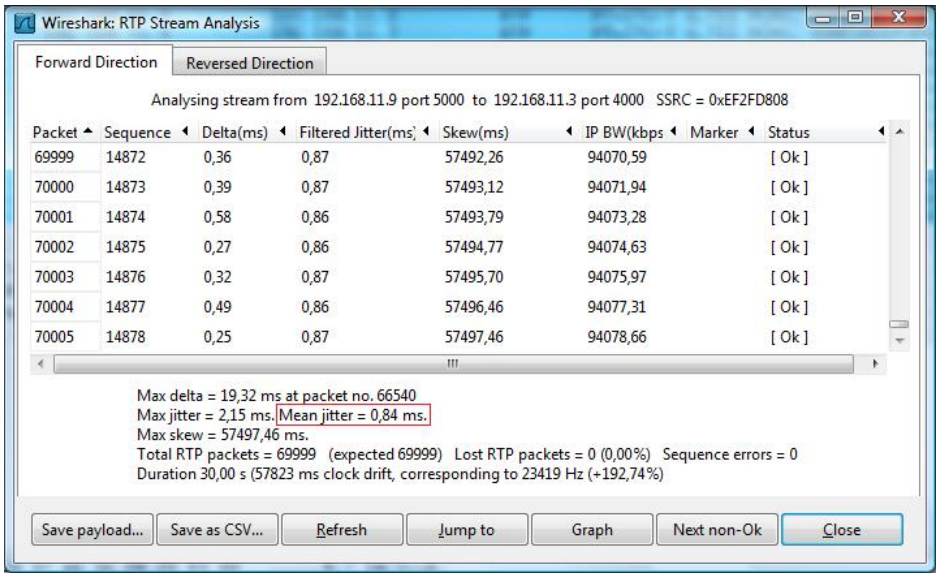

Figure 4.10: WireShark RTP stream analisys for LAN measurements

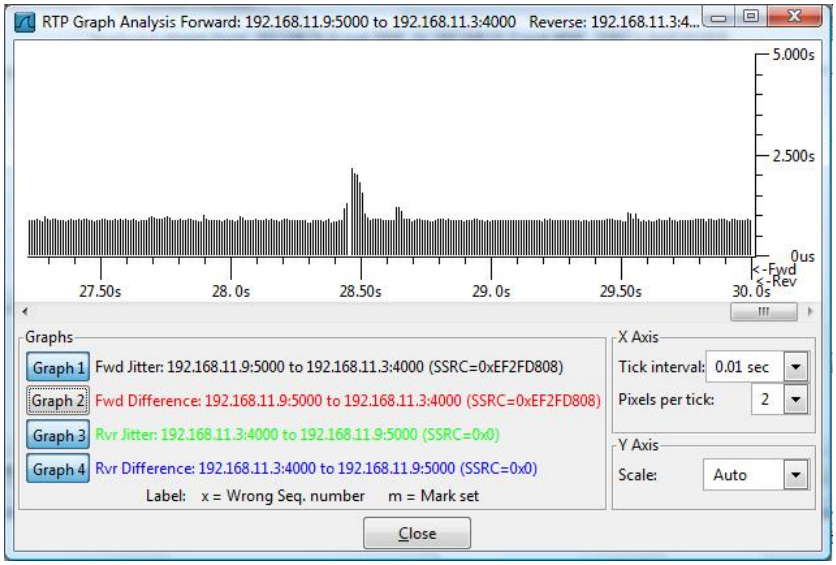

Figure 4.11: Interarrival jitter graph for LAN measurements

Chapter 5

Conclusions and Future Work

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"*
Andrs Lass, Tams Urbancsek, dm Helybly.
Budapest University of Technology and Economics Department of Control Engineering and Information Technology

[2] *"Jrtplib online documentation"*
http://research.edm.uhasselt.be

[3] *"Il protocollo di trasporto RTP"*
http://fly.isti.cnr.it/

[4] *"RFC 3550"*
http://tools.ietf.org

[5] *"RTP performances metrics"*
http://www.ietf.org

[6] *"RTP protocol interarrival jitter RFC 1889"*
http://www.faqs.org/rfcs

[7] *"Voice packet interarrival jitter over IP switching"*
Tomi Yletyinen and Raimo Kantola.
Helsinki University of Technology, Laboratory of Telecommunications Technology

[8] *"WireShark network protocol analyzer,"*
http://www.wireshark.org

[9] *OpencvWiki*
http://opencv.willowgarage.com/wiki/