ON THE IMPLEMENTATION OF SERVERS FOR LARGE SCALE CAR SYSTEMS BASED ON MOBILE PHONES

Víctor Fernández[1], Juan Manuel Orduña[1] and Pedro Morillo[1]
[1]*Departamento de Informática, University of Valencia, Valencia, Spain*
{*Victor.Fernandez-Bauset, Juan.Orduna,Pedro.Morillo*}*@uv.es*

Keywords: Collaborative Augmented Reality, Mobile Phones.

Abstract: Collaborative Augmented Reality (CAR) systems allow multiple users to share a real world environment including computer-generated images in real time. The hardware features of most current mobile phones include wireless network capabilities that offer a natural platform for CAR systems. However, the potential number of clients in CAR systems based on mobile phones is much larger than on CAR systems based on other kind of mobile devices, requiring a system design that takes into account scalability issues. This paper presents the experimental comparison of different CAR systems based on mobile phones with different server implementations. The performance evaluation results show that the best implementation is the one based on UDP messages instead of classical TCP connections, in order to improve the system throughput. The UDP-based implementation provides a significant improvement in system throughput, at the cost of loosing a very small percentage of updating messages. However, the effects of these small quantities of dropped messages cannot expand beyond some jitter (bounded within a short period of time) in a reduced number of clients of the CAR application. These results validate the proposed UDP-based implementation as the best option for large-scale CAR systems based on mobile phones.

1 INTRODUCTION

Augmented Reality (AR) systems are nowadays widely used in applications such as medical procedures, scientific visualization, manufacturing automation, cultural heritage and military applications. The term Augmented Reality (AR) refers to computer graphic procedures or applications where the real-world view is superimposed by computer-generated objects in real-time (Azuma, 1997; Azuma et al., 2001; Cawood and Fiala, 2008). From the beginning of AR systems, the potential of collaborative AR (CAR) systems was exploited for different activities such as Collaborative Computing (Billinghurst et al., 2000) or Teleconferencing (Billinghurst and Kato, 1999). Wearable devices were used to provide CAR systems, where a wearable AR user could collaborate with a remote user at a desktop computer (Hallerer et al., 1999; Piekarski and Thomas, 2002).

On other hand, a lot of different mobile and/or wereable devices comprising a computing embedded system pervade our daily life, and they have been used for CAR systems. Mobile phones have become the most extended example of these devices (Henrysson and Ollila, 2004; Mahring et al., 2004). They have become an ideal platform for CAR systems, due to the multimedia hardware they include, like full color displays, integrated cameras, fast processors and even dedicated 3D graphics chips (Henrysson et al., 2005).

As an example, Figure 1 shows a CAR system developed for collaborative training in industrial electricity. In this case, the CAR systems show the electrical technicians how the circuit-breakers should be replaced in the electric general panelboards at the construction sites. The figure shows on the left image the execution of the CAR tool on a Samsung Galaxy NOTE mobile phone. The image on the center shows a real image of the the panelboard where technicians collaboratively operate, and the right image shows the execution of the CAR tool on a HTC Nexus One mobile phone.

Nevertheless, the wide variety of current mobile phones, with different graphic and processing capabilites, and different operating systems, can have significant effects on the performance of a large-scale CAR system, in terms of system latency, frames per second or number of supported clients with certain latency levels. These effects should be taken into account when implementing CAR systems based on mobile phones, in order to avoid a performance degra-

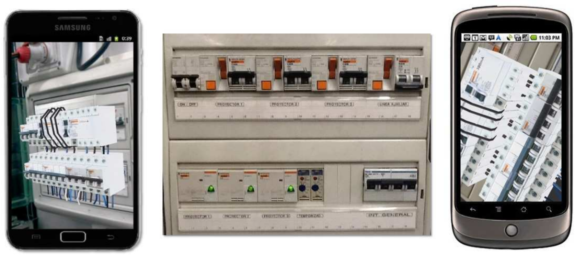

Figure 1: Example of a CAR application developed for training in industrial electricity.

dation in terms of both system latency and throughput.

In previous works, we have characterized the behavior of different mobile phones when used in Collaborative Augmented Reality applications, (Bauset et al., 2011). Also, we have carried out a performance characterization from the server side, measuring the system response time and system throughput when varying different systems parameters like the number of clients in the system, the number of clients in the work space (i.e., the number of neighbors to which messages should be sent), and the cycle time of clients (Bauset et al., 2012). The characterization results shows that the system saturation point depends on the overall percentage of CPU utilization in the computer platform acting as the system server. Although the CPU threshold is not a fixed value, it is inversely related to the number of processor cores. The results also showed that the CAR systems throughput heavily depends on the kind of client devices, but for certain kind of devices, the system bottleneck is the server I/O.

In this paper, we propose a comparative study of different implementations of the CAR server, in order to improve the performance of CAR systems based on mobile phones. The performance evaluation results show that the implementation providing the best performance is the one based on UDP messages. The UDP-based implementation provides a significant improvement in system throughput with respect to other implementations based on TCP, at the cost of loosing a very small percentage of updating messages. However, the effects of these small quantities of dropped messages cannot expand beyond some jitter (bounded within a short period of time) in a reduced number of clients of the CAR application. These results validate the proposed UDP-based implementation as the best option for large-scale CAR systems based on mobile phones.

The rest of the paper is organized as follows: Section 2 shows some related work about CAR applications on mobile phones. Section 3 describes the different CAR implementations considered for comparison purposes, and Section 4 shows the performance evaluation results. Finally, Section 5 presents some conclusion remarks and the future work to be done.

2 RELATED WORK

Augmented Reality superimposes multimedia content - 3D object, text, sound, etc - to real world through a display or screen. In order to locate digital contents on a specific image of the real world point, some references within the image is needed. These references are known as markers, and two methods are usually used: natural feature tracking and fiducial marker tracking. The former method uses interest point detectors and matching schemes to associate 2D locations on the video with 3D locations (Wagner et al., 2008). This process can be grouped in three big phases: interest point detection, creation of descriptor vectors for these interest points, and comparison of vectors with the database (Lee et al., 2009). The latter method uses fiducial markers to find a specific position of real world. This process can be divided in three phases: edge detection, rejection of quadrangles that are too large or too small, and checking against the set of known patterns (Wagner et al., 2008).

Any CAR application needs a device equipped with an on-board camera, CPU and display. The most common devices used for CAR applications are Tablet-PCs or mobile phones. We will focus on mobile phones, because they are more suitable for CAR applications (Henrysson et al., 2005; Thomas, 2007).

There are few solutions based on fiducial marker tracking over mobile phones. In 2003, ArToolKit (Kato, 2011), one of the most well-known software libraries for developing Augmented Reality (AR) application, was released for Windows CE, and the first self-contained application was developed for mobile phones (Wagner and Schmalstieg, 2003). This software evolved later as the ArToolKitPlus tracking library (Wagner et al., 2008). A tracking solution for mobile phones that works with 3D color-coded marks was developed (Mahring et al., 2004), and a version of ArToolKit for Symbian OS was developed, partially based on the ArToolKitPlus source code (Henrysson et al., 2005). The research teams behind these works have worked on fiducial marker tracking, but not from the collaborative point of view. Also, there are many other works that focus on natural feature tracking (Wagner et al., 2008; Srinivasan et al., 2009; Wagner et al., 2010; Wagner et al., 2009).

Although real-time natural feature tracking over mobile devices has been currently achieved (Wagner et al., 2008), fiducial marker tracking is more widely used, because it allows simultaneous computational robustness and efficiency. A large number of locations and objects can be efficiently labeled by encoding unique identifiers on the markers. Additionally, the markers can be detected with angles near to 90 degrees (Wagner et al., 2008).

The first CAR applications improved the conference system highlights, giving the feeling of real presence to remote collaborators (Billinghurst and Kato, 1999). The Rekimoto's Transvision system showed how to share virtual objects through handheld displays (Rekimoto, 1996). Also, Schmalstieg created a software architecture to develop CAR applications (Szalavri et al., 1998).

3 SERVER IMPLEMENTATIONS

In order to analyze the behavior of CAR systems based on mobile devices, we have developed a multithreaded CAR server that supports simulated clients (simulated mobile devices) with the behavior measured in our previous work (Bauset et al., 2011). We have time-stamped every message generated within this CAR system, in order to measure the performance of every device. The system configuration will consist of one server, and a certain amount of mobile devices that are scanning the visual space of their video camera looking for a marker that will be converted into a 3D object in their display. The main performance metrics in distributed systems are throughput and latency (Duato et al., 1997). However, in order to avoid clock skews when measuring the system latency in distributed systems, the same device should measure the initial and final time. Therefore, we consider round-trip times instead of system latencies.

Since we are considering collaborative systems, after each updating of the object location, the mobile device will send a location update message (containing the new location) to each of its neighbor devices. The neighbor devices are those who participates in the same collaborative task, and we have denoted this set of neighbor devices as a *working group*. An important parameter of the system configuration will be the working group size, since it determines the amount of location update messages to be exchanged in each cycle). The messages are sent through the server (that is, it sends the location update message to the server, and then the server re-sends the message to the appropriate clients). For performance evaluation purposes, the destination clients return an acknowledgment message (ACK) to the server, which, in turn, forwards it to the source client. When the source client has received the ACK messages corresponding to the location update from all the clients in its working group, then it computes the average system response for that location update. Figure 2 illustrates the action cycle that takes place for each of the mobile clients in the system.

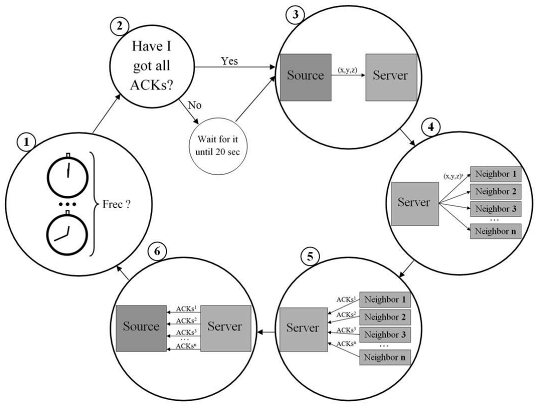

Figure 2: Stages of the action cycle in each mobile device.

Once the message with the location update is sent, the action cycle performed by each client is composed of the following steps: first, it performs one new image acquisition followed by a marker detection stages. Then, the client waits until the cycle period (determined by the action frequency, a system parameter) finishes. Next, if the acknowledgments from all the neighbors have been received, a new message with the new marker location is sent. If not all the acknowledgments have been received, then it waits until a maximum threshold of 20 seconds, and then a new round of messages (with the latest marker location) are sent to the neighbors through the server. The neighbors simply returns an ACK message to the sender device through the server. The server simply forwards the messages to the corresponding destination clients. It must be noticed that the mobile devices will not send a new round of messages with a new location update until it has received the acknowledgment message from all its neighbors, even although new marker detection stages have been completed in the device.

This characterization setup considers that all the required static content in the scene has been loaded. According to recent works (Kantonen, 2009), in these cases the network bandwidth required is less than 50 kbps for performing this information exchange. Since we are using a Gigabit Ethernet, we ensure that network bandwidth does not become a system bottleneck.

The system latency provided for each location update is computed by recording a timestamp when the first message is sent to the server. Next, a second timestamp is recorded with the last ACK message for that location update received from the server. The system response time is computed by subtracting these two timestamps. The server response time is computed by timestamping both each message forwarded from each client and the reception of the corresponding ACK message from the destination client. Also, the percentage of CPU utilization is measured both in the server and the mobile devices every half second.

On other hand, each client process simulates 50 mobile devices, using two threads per simulated device. We have uniformly distributed the number of the required client processes for each system configuration. Since we have 10 desktop computers available for hosting the clients, the configuration for 1000 clients consists of 10 computers hosting 5 client processes each (100 threads per computer hosting clients). We have experimentally ensured that none of the computers hosting clients is close to saturation by measuring the average time required for answering the messages sent to the clients hosted in each computer.

The previous work showed that Google phone HTC Nexus One was the fastest device, with a period cycle of 167.11 milliseconds, while the Motorola Milestone was the slowest one, with a period cycle of 698.34 milliseconds(Bauset et al., 2011). We have considered these values as the limits for characterization purposes. Also, we have considered four different values for the working group size: 5, 10, 20, and 25 neighbor clients. Finally, we have considered a number of clients in the system ranging from 100 to 1000. It must be noticed that usually, actual CAR applications do not contain more than a hundred clients (for example, more than a hundred persons within the same lounge using collaborative Augmented Reality for studying art masterpieces), due to the size of the augmented models. Thus, reaching thousands of clients clearly exceed the worst case for this kind of applications.

We have implemented a multithreaded server, where each server thread manages a group of clients within a given working group (i.e., the people involved in the same collaborative task). Thus, for example, with a system configuration of 500 mobile clients and an working group size of 10 clients, we have 50 server threads (50 working groups of 10 people each), and each thread supports 10 clients. We have considered a maximum configuration of 1000 clients, resulting in 100 server threads. For comparison purposes, we have considered a single server. Nevertheless, the system performance greatly depends on the server implementation, and we have considered three different server implementations:

3.1 TCP Implementation

The simulator starts generating a *Server Process*, and for every 50 clients it generates a *Client Process*. Figure 3 illustrates the general scheme of the Server Process. This process starts listening connections, and for each connection it generate a new array of *X* TCP sockets, where *X* is the number of clients that will be within a given working group. When all the clients have connected to the Server Process (the population size is a simulation parameter) then the Server Process generates as many *Server Threads* as needed. Each Server Thread is in charge of managing all the clients within a working group. Concretely, it starts the simulation by sending a welcome message to all the client sockets. When the simulation finishes, it collects statistics from all the clients in its working group. But the most important task performed by server threads is the generation of two threads for each of the clients in the working group: the *Server Receiver Thread (SRT)* and the *Server Processor Thread (SPT)*. The SRT associated to client *i* receives the location update messages from the client *i*, it computes the correct destination clients (the neighbor clients, that is, the clients within the same working group) and it generates messages that will be stored in the queues of the Server threads managing these neighbor clients. The SPT associated to client *i* extracts the queued messages that the SRTs associated to other clients may have generated for client *i*, and it sends them to this client. Additionally, the server process collects and processes the statistics generated by the server threads, and it also measures the percentage of CPU utilization.

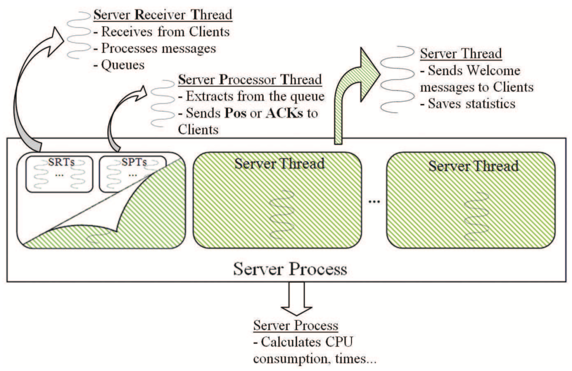

Figure 3: General scheme of the server process in the TCP implementation.

Figure 4 illustrates the general scheme of the Client Process. This process generates 50 client threads (we have assumed a maximum population size of 1000 client devices), and it also computes the percentage of CPU utilization, client latencies, etc.. Each Client Thread generates two threads for each client: the *Client Receiver Thread (CRT)* and the *Client Processor Thread (CPT)*, and when the welcome message from the Server Thread arrives to the associated socket, then the Client Thread starts the simulation, that consists of sending a given number of position update messages and receiving the corresponding acknowledgments from the neighbor clients. The frequency of the location update messages is a simulation parameter (it determines the actuation rate of clients to be simulated). In each simulation cycle (that can be much shorter than the client actuation cycle), the Client Thread checks if a new location update message should be sent. If so, it then checks if all the acknowledgments of the previous message have arrived from the neighbors clients. If not, then it waits for them until a maximum timeout of 20 seconds. The value for this timeout has been empirically obtained. Although it is not shown here for the sake of shortness, we have performed experiments with the simulator, concluding that unless the system reaches deep saturation and collapses, the maximum latency for obtaining all the acks from the neighbors have been 20 seconds. After this timeout, the new location update is send. The CRT is continuously checking the client queue. When a location update message arrives to this queue, it sends back an acknowledgment to the corresponding server thread. If an acknowledgment of a previous message arrives to this queue, this acknowledgment is computed.

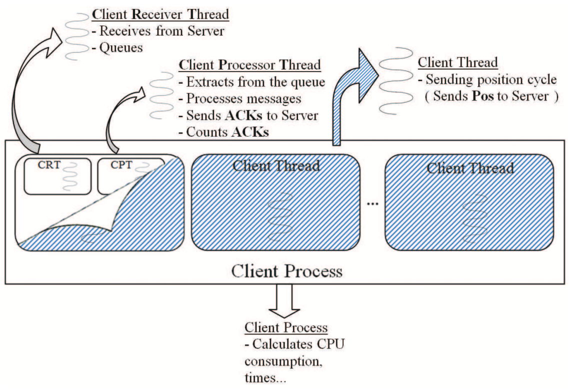

Figure 4: General scheme of the client process in the TCP implementation.

3.2 TCP-Select Implementation

One of the potential limitations of the TCP implementation is the server overhead due to the huge number of threads, for those cases when the population size increases. In order to reduce this overhead, we have carried out a different server implementation that, although it is also based on TCP connections, it uses the `Select` function of BSD sockets (Jones, 2003).

In this server implementation, each Server Thread has a single SRT and a single SPT for managing all the clients in each working group, instead of one SRT and one SPT for each client. Using the `Select` function, the SRT receives messages from all the clients and it processes them. We have tested different options regarding the best number of SPTs for managing all the clients in each working group. Although they are not shown here for the sake of shortness, the experimental results showed that no significant improvements were achieved when using more than a single SPT for managing, in terms of the obtained latencies.

3.3 UDP Implementation

Finally, we have considered a connectionless oriented implementation for the CAR system, in order to study the effectiveness of TCP connections in a distributed environment like a CAR system. The motivation of this study are both the short message size (usually carry a position update consisting of a bunch of bytes) and the huge amount of the messages generated by CAR systems. For this kind of frequent but short network traffic, usually connectionless oriented protocols show a better network performance (Duato et al., 1997). Although the UDP protocol can loose messages and the effects and size of these losses should be studied, we have also considered this implementation for comparison purposes.

The UDP implementation is very similar to the TCP-Select implementation. The only difference is that in this implementation we have used UDP sockets. Since this implementation can drop messages, it also counts the number of dropped or lost messages (since both the number of iterations and the number of clients in each working group is known, each client can compute the number of message that should arrive).

4 PERFORMANCE EVALUATION

This section shows the performance evaluation of the implementations described in the previous section. We have performed different measurements on different simulated systems using these implementations. Like other distributed systems, the most important performance measurements in CAR systems are latency and throughput (Duato et al., 1997). Since we are focusing on large scale CAR systems, we have performed simulations with different number of clients and we have measured the response time provided to these clients (the round-trip delay for each updating message sent by a given client to the clients in its working group). In this way, we can study the maximum number of clients that the system can support while providing a response time below a given threshold value. In order to define an acceptable behavior for the system, we have considered 250 ms. as the threshold value, since it is considered as the limit for providing realistic effects to users in DVEs (Henderson and Bhatti, 2003).

In order to evaluate the performance provided by each of the proposed implementations, we have carried out simulations and we have measured the average system response for all the location updates send by all the clients in the system. In this sense, we have considered the system response time (in milliseconds) for each updating message sent by a given client to its neighbor clients as the time required for receiving the acknowledgments from all the clients in the working group of this given client. In order to measure the dispersion of this metric, we have measured the standard deviation for all the updating messages sent, as well. Also, we have computed the response time in the server (in milliseconds) as the time required by the destination clients to answer the server messages. We have measured both the average and the maximum values measured in the server for each simulation. Additionally, we have computed the percentage of the CPU utilization in the system server, since it can easily become the system bottleneck. The computer platform hosting the system server is a Intel Core 2 Duo E8400 CPU running at 3.00 GHz with 4 Gbytes of RAM, executing an Ubuntu Linux distribution with the 3.0.0-14-generic x86 64 operating system kernel.

In order to study the system behavior for different levels of workload, we have repeated simulations with working group sizes of 10,15,20 and 25 clients. Although not all of them are shown here for the sake of shortness, we show the result for the smallest (5 clients in each working group) and the biggest sizes (25 clients in each working group).

Table 1 shows the results for a CAR system whose client devices are all of them Nexus One, and where the working group size for each client is of five neighbor clients. This table shows the results for the three considered implementations, organized as three subtables with ten rows each, and labeled with the name of the implementation (TCP, TCP-Select and UDP). The most-left column in these subtables shows the number of clients in the system, that is, the population size. The values in this column range from 100 to 1000 clients in the system. The next two columns show the average value of the response times (in milliseconds) provided by the system to all the clients (labeled as ”RT”), as well as the corresponding standard deviation values (column labeled as ”Dev”). The fourth column (labeled as ”CPU”) shows the percentage of the CPU utilization in the server. Finally, the fifth and sixth columns (labeled as ”RT_S” and ”RT_SM”, respectively) show the average and maximum values (in milliseconds) of the response time in the server for all the messages exchanged during the simulation.

	TCP implementation				
Size	RT	Dev	CPU	RT_S	RT_SM
100	62.37	22.68	9.9	19.36	20.9
200	63.77	22.21	15	20.12	22.43
300	66.71	22.66	22	25.15	28.28
400	68.68	22.5	32.7	27.14	29.56
500	71.04	23.56	45	27.14	30.9
600	71.5	24.18	48.6	26.58	31.95
700	72.37	25.01	59	27.17	35.42
800	72.85	26.01	68	27.87	34.54
900	75.01	28.98	79.2	28.61	35.89
1000	147.33	101.71	85	43.95	48.66
	TCP-Select implementation				
Size	RT	Dev	CPU	RT_S	RT_SM
100	65.77	21.56	8	20.44	24.96
200	67.12	22.71	11.2	21.5	23.54
300	67.52	22.6	19.8	23.67	26.62
400	67.64	22.88	28	23.21	27.28
500	69.12	23.23	31	25.31	29.34
600	69.14	23	39.6	25.28	28.74
700	69.37	23.45	47	24.53	30.72
800	75.75	26.63	54.5	26.96	35.16
900	70.24	24.81	59.6	24.02	31.87
1000	71.05	27.53	67	22.64	35.04
	UDP implementation				
Size	RT	Dev	CPU	RT_S	RT_SM
100	4.80	7.06	38.40	1.95	3.22
200	3.76	4.95	34.70	1.57	2.84
300	9.57	9.74	26.00	4.44	10.97
400	3.60	5.18	33.70	1.46	3.16
500	4.59	6.41	41.60	1.81	3.77
600	7.34	13.17	47.00	3.23	17.16
700	5.28	8.24	53.00	2.11	7.76
800	7.10	18.52	84.10	2.65	16.53
900	5.85	11.69	66.30	2.61	12.05
1000	7.15	15.47	69.50	2.87	15.18

Table 1: Results for a working group size of 5 neighbors

Table 1 that none of the values in the RT column reaches the threshold value of 250 milliseconds in any of the considered implementations, showing that the system can efficiently support up to one thousand clients while interactively displaying the Augmented Reality. Nevertheless, there are significant differences in this column among the considered implementations. Thus, the TCP implementation shows a huge rise in the response time when the system reaches one thousand clients, passing from around 75 milliseconds to more than 147 milliseconds as an average. The standard deviation of this values is also more than three times the value shown for nine hundred clients. These values show that for that population size the system is approaching saturation. On the contrary, the TCP-Select implementation does not show an increase in neither the column RT nor the column Dev for a population of one thousand clients. Moreover, the UDP implementation shows RT values that are one order of magnitude lower than the ones shown by the other two implementations.

The third column in table oneAOI5 shows that the CPU utilization increases as the number of clients in the system increases. In the case of the TCP implementation, the system approaches saturation when the server reaches 85% of CPU utilization. For lower percentages of CPU utilization the response times do not significantly increase. It is worth mention than the UDP implementation provides RT values of one order of magnitude lower even for CPU utilization of around 70%. These values show that the latency provided by CAR systems greatly depends on the connection or connectionless scheme followed by the system to exchange information with the clients.

Finally, the columns RT_S and RT_SM show that most of the response time provided to clients is due to processing in the server. Thus, for example, the results for the TCP implementation and a system size of 900 clients show that as an average each client has to wait 75.01 milliseconds for receiving the acknowledgments from all the clients in its working group, but as an average the server must wait only 28.61 milliseconds to receive answers from clients. This difference highly increases for the case of one thousand clients, where the response time obtained by the server from clients is around 44 milliseconds but the average response time provided to clients is 147.33, around three times higher. It is also worth mention that the ratio between the RT_S and the RT columns do not significantly vary among the three implementations. Finally, the RT_SM column shows that the maximum values in the RT_S parameter do not exceed the value in the RT column for both TCP implementation, and they do not exceed twice the value in the RT column of the UDP implementation. Therefore, we can conclude that most of the time required to acknowledge each client update is due to the processing of the updates and acknowledgments in the server.

These results show that the best latencies when the system is far from saturation are provided with the UDP implementation. However, UDP is a connectionless-oriented protocol, and as such, it may drop messages when the system approach saturation. In order to study these effects, as well as the general behavior of the system for a high workload, table 2 shows the results for a working group size of 25 clients. In this table, the column labeled as ”RT_SM” has been removed, since the results were similar to those shown in table 1. Instead, we have added a new column, labeled as ”% loss”, that shows the percentage of messages dropped by the system. It has been computed by subtracting the number of messages received by all the clients in a simulation (measured in the simulation itself) from the theoretical number of messages that clients should exchange for a given population size.

	TCP implementation				
Size	RT	Dev	CPU	RT_S	% loss
100	96.03	25.91	60.4	24.26	0.00
200	103.19	38.71	83	30.98	0.00
300	167.58	50.19	86.1	40.24	0.00
400	250.53	77.68	85	49.89	0.00
500	357.13	126.19	92	74.41	0.00
600	496.62	218.79	94.1	96.25	0.00
700	480.87	156.16	85.3	93.42	0.00
800	531.93	210.02	86	97.47	0.00
900	505.89	300.11	89	93.45	0.00
1000	524.96	372.62	87.1	82.39	0.00
	TCP-Select implementation				
Size	RT	Dev	CPU	RT_S	% loss
100	90.8	24.7	23.2	19.35	0.00
200	89.95	21.13	47	33.4	0.00
300	123.95	32.36	72	54.7	0.00
400	209.2	35.88	87.2	85.55	0.00
500	268.17	44.44	86	112.07	0.00
600	331.41	50.55	87	143.62	0.00
700	383.96	70.6	93.1	151.56	0.00
800	454.57	125.21	90.1	151.56	0.00
900	491.59	106.18	91	147.56	0.00
1000	566.44	133.33	93.1	166.79	0.00
	UDP implementation				
Size	RT	Dev	CPU	RT_S	% loss
100	9.86	6.78	72.50	4.06	0.83
200	21.70	14.73	82.00	9.84	1.18
300	26.01	21.91	79.60	11.61	0.69
400	39.41	30.66	81.90	18.26	0.83
500	48.68	39.68	83.80	22.84	0.74
600	62.83	45.16	83.80	28.96	0.74
700	79.70	97.87	85.10	37.26	0.76
800	87.18	200.95	84.00	40.20	1.04
900	93.64	66.36	83.90	40.71	0.93
1000	122.37	85.35	85.00	44.98	0.90

Table 2: Results for a working group size of 25 neighbors

Table 2 shows that for this level of workload the system enters saturation in the two TCP-based implementations. Effectively, the RT column shows that the TCP implementation reaches the maximum threshold value of 250 ms. with 400 clients. From that population size up, the response time provided to clients is unacceptable. The TCP-Select implementation reaches (and exceeds) this threshold value for a population of 500 clients. However, the UDP implementation does not reach even half of this value for the maximum population size considered, one thousand clients.

It is worth mention that for those cases when the system reaches saturation, the percentage of CPU utilization in the server is 85% or higher. The gap between 85% and 98% of CPU utilization for reaching the saturation point can be explained by the shared memory architecture of current multicore processors (the dual core processor in the computer platform used as simulation server), as shown in (Bauset et al., 2012). The synchronization of the kernel calls, together with the synchronization among threads in the application, prevent the CAR system from fully exploiting the computational power of all the processing cores at the same time, reaching saturation for a lower overall percentage of CPU utilization. The more processor cores in the processor, the higher percentage of CPU utilization ”wasted” in synchronization.

The ”% loss” column shows that for the UDP implementation the percentage of lost messages is not higher than 1.2%. The effects of loosing some messages will consists of delaying the location update of the artificial artifacts in the display of the destination clients, that is, they will produce some jitter in the display of the clients. However, these percentage values ensure a reasonable quality in the visualization of the CAR system. Therefore, we can state that the UDP implementation provides the highest throughput and the best latency for large scale CAR applications, at the cost of a very limited effects on the visualization in some clients.

In order to ensure that the effects of the UDP implementation in terms of dropped messages are consistent for all the workload levels considered, figure 5 shows the average number of packets dropped for each working group size considered.

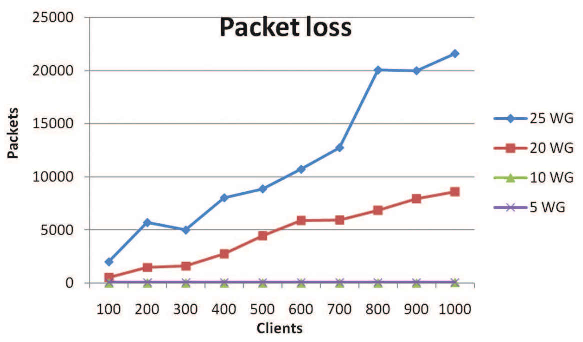

Figure 5: Number of packets lost in the UDP implementation.

Figure 5 shows that for working group sizes of 5 and 10 neighbor clients there are no packet losses. For a working group size of 20 neighbors, the amount of lost packets reaches 8581 for a theoretical total of packets sent of 1.9 million packets. Analogously, for a working group size of 25 neighbors, the amount of lost packets reaches 21593 out of 2.4 million packets sent. Therefore, in the worst case the number of lost packets only represent a 1’18 % of the total amount of packets sent. This represents only a small image flicker on some clients, and in very limited periods of time. As the information is sent more than once per second - remember the action cycle of the Nexus One is 167.11 ms - this can be considered insignificant flickering.

Although they are not here for the shake of shortness, we repeated the same tests shown in this section here using a different client device, the Motorola Milestone, and we obtained analogous results. Those results were less interesting because of the bigger action cycle of the Milestone (698.34 ms). With that action frequency the system saturation point was not reached even in the worst case of a working group size of 25 neighbors and a population of 1000 clients. We have shown here the results for the Nexus One as the worst case for the server implementation.

5 CONCLUSIONS

This paper has proposed the experimental comparison of different large-scale CAR systems based on mobile phones with different server implementations. The performance evaluation results show that the best implementation is the one based on UDP messages, instead of classical TCP connections, in order to improve the system thoughput. The UDP-based implementation provides a significant improvement in both system throughput and system response time, at the cost of loosing a very small percentage of updating messages. However, the effects of these small quantities of dropped messages cannot expand beyond some jitter (bounded within a short period of time) in a reduced number of clients of the CAR application. These results validate the proposed UDP-based implementation as the best option for large-scale CAR systems based on mobile phones.

ACKNOWLEDGEMENTS

This work has been jointly supported by the Spanish MICINN and the European Commission

FEDER funds, under grants Consolider-Ingenio 2010 CSD2006-00046 and TIN2009-14475-C04.

REFERENCES

Azuma, R. (1997). A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385.

Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., and MacIntyre, B. (2001). Recent advances in augmented reality. *Computer Graphics and Applications, IEEE*, 21(6):34 –47.

Bauset, V. F., Orduña, J. M., and Morillo, P. (2011). Performance characterization on mobile phones for collaborative augmented reality (car) applications. In *Proceedings of the 2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications*, DS-RT '11, pages 52–53, Washington, DC, USA. IEEE Computer Society.

Bauset, V. F., Orduña, J. M., and Morillo, P. (2012). On the characterization of car systems based on mobile computing. In *The Fifth INTERNATIONAL SYMPOSIUM ON ADVANCES OF HIGH PERFORMANCE COMPUTING AND NETWORKING (AHPCN-2012)*, Proceedings of the 14th IEEE International Conference on High Performance Computing and Communications (HPCC '12), Washington, DC, USA. IEEE Computer Society.

Billinghurst, M. and Kato, H. (1999). Real world teleconferencing. In *Proc. of the conference on Human Factors in Computing Systems (CHI 99)*.

Billinghurst, M., Poupyrev, I., Kato, H., and May, R. (2000). Mixing realities in shared space: an augmented reality interface for collaborative computing. In *IEEE International Conference on Multimedia and Expo (ICME 2000)*, volume 3, pages 1641–1644.

Cawood, S. and Fiala, M. (2008). *Augmented Reality: A Practical Guide*. Pragmatic Bookshelf.

Duato, J., Yalamanchili, S., and Ni, L. (1997). *Interconnection Networks: An Engineering Approach*. IEEE Computer Society Press.

Hallerer, T., Feiner, S., Terauchi, T., Rashid, G., and Hallaway, D. (1999). Exploring mars: Developing indoor and outdoor user interfaces to a mobile augmented reality system. *Computers and Graphics*, 23:779–785.

Henderson, T. and Bhatti, S. (2003). Networked games: a qos-sensitive application for qos-insensitive users? In *Proceedings of the ACM SIGCOMM 2003*, pages 141–147. ACM Press / ACM SIGCOMM.

Henrysson, A., Billinghurst, M., and Ollila, M. (2005). Face to face collaborative ar on mobile phones. In *Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on*, pages 80 – 89.

Henrysson, A. and Ollila, M. (2004). Umar: Ubiquitous mobile augmented reality. In *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, MUM '04, pages 41–45, New York, NY, USA. ACM.

Jones, M. T. (2003). *BSD Sockets Programming from a Multi-Language Perspective*. Charles River Media, Inc., Rockland, MA, USA.

Kantonen, T. (2009). Augmented collaboration in mixed environments. Master's thesis, Helsinky University of Technology.

Kato, D. H. (2011). Artoolkit. Available at http://www.hitl.washington.edu/artoolkit/.

Lee, S. E., Zhang, Y., Fang, Z., Srinivasan, S., Iyer, R., and Newell, D. (2009). Accelerating mobile augmented reality on a handheld platform. In *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, pages 419 –426.

Mahring, M., Lessig, C., and Bimber, O. (2004). Video see-through ar on consumer cell-phones. In *ISMAR'04*, pages 252–253.

Piekarski, W. and Thomas, B. H. (2002). Tinmith-hand: Unified user interface technology for mobile outdoor augmented reality and indoor virtual reality.

Rekimoto, J. (1996). Transvision: A hand-held augmented reality system for collaborative design. In *Virtual Systems and Multi-Media (VSMM)'96*.

Srinivasan, S., Fang, Z., Iyer, R., Zhang, S., Espig, M., Newell, D., Cermak, D., Wu, Y., Kozintsev, I., and Haussecker, H. (2009). Performance characterization and optimization of mobile augmented reality on handheld platforms. In *Workload Characterization. IISWC 2009. IEEE International Symposium on*, pages 128 –137.

Szalavri, Z., Schmalstieg, D., Fuhrmann, A., and Gervautz, M. (1998). studierstube: An environment for collaboration in augmented reality. *Virtual Reality*, 3:37–48.

Thomas, M. H. M. B. B. (2007). *Emerging Technologies of Augmented Reality: Interfaces and Design*. IGI Global.

Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., and Schmalstieg, D. (2008). Pose tracking from natural features on mobile phones. In

Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, ISMAR '08, pages 125–134, Washington, DC, USA. IEEE Computer Society.

Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., and Schmalstieg, D. (2010). Real-time detection and tracking for augmented reality on mobile phones. *Visualization and Computer Graphics, IEEE Transactions on*, 16(3):355 –368.

Wagner, D. and Schmalstieg, D. (2003). First steps towards handheld augmented reality. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers*, ISWC '03, pages 127–135, Washington, DC, USA. IEEE Computer Society.

Wagner, D., Schmalstieg, D., and Bischof, H. (2009). Multiple target detection and tracking with guaranteed framerates on mobile phones. In *Mixed and Augmented Reality. ISMAR 2009. 8th IEEE International Symposium on*, pages 57 –64.