Jurnal Teknologi

Full paper

Transfering Data from a Server to an Android Mobile Application: A Case Study

Ricardo Anacleto[a*], Lino Figueiredo[a], Ana Almeida[a], Paulo Novais[b]

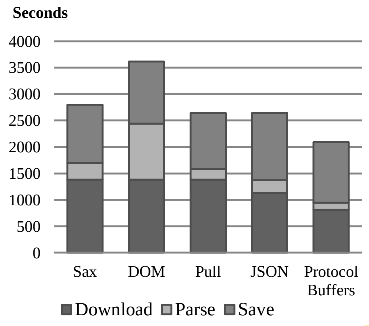
[a]*GECAD–Knowledge Engineering and Decision Support Research Center, at School of Engineering of the Polytechnic Institute of Porto, Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto, Portugal*
[b]*CCTC–Computer Science and Technology Center, at University of Minho, Campus de Gualtar, 4710-057 Braga, Portugal*

*Corresponding author: rmsao@isep.ipp.pt

Article history

Received :11 September 2012
Received in revised form :
21 February 2013
Accepted :15 April 2013

Graphical abstract



Abstract

Nowadays, due to the incredible grow of the mobile devices market, when we want to implement a client-server applications we must consider mobile devices limitations. In this paper we discuss which can be the more reliable and fast way to exchange information between a server and an Android mobile application. This is an important issue because with a responsive application the user experience is more enjoyable. In this paper we present a study that test and evaluate two data transfer protocols, socket and HTTP, and three data serialization formats (XML, JSON and Protocol Buffers) using different environments and mobile devices to realize which is the most practical and fast to use.

Keywords: Client-server communication; data serialization; parsers; mobile applications; performance; protocol buffers; algorithms

■1.0 INTRODUCTION

In the last years, mobile applications development had an incredible grow ratio, and in this large spectrum some of them need to connect to an already developed server application. This type of integration can be done in many different ways. Therefore the task of develop a mobile solution can often be daunting considering all the technology choices and implementation approaches available. Unlike centralized systems, mobile applications need software optimization. These optimizations need to consider the network traffic consumption, battery consumption and system responsiveness.

It's clear that current mobile devices still have several limitations when compared to traditional computers. It was based on these limitations that led us to the question: Which is the best way to exchange information between a server and a mobile client in order to minimize these limitations?

This question started to appear when we were developing the PSiS (Personalized Sightseeing Planning System) Mobile [1], which is a mobile application intended to support a tourist when he is on vacations, recommending points of interest and helping planning his stay - more information about PSiS Mobile is presented on section 2.

In sections 3 and 4 different data transfer protocols and data serialization methods are presented, as well as, the results for this case study, which involves the transfer of points of interest from the server's database into the mobile device database using the previous presented technologies. Based in some metrics we evaluate the results and realize which protocol and serialization format are the more appropriate to use. Finally, in section 5 we present the time that it takes to save data to a mobile database, as well as, the overall time that download, parse and save operations take. Also, we analyze and discuss the obtained results and point out some conclusions.

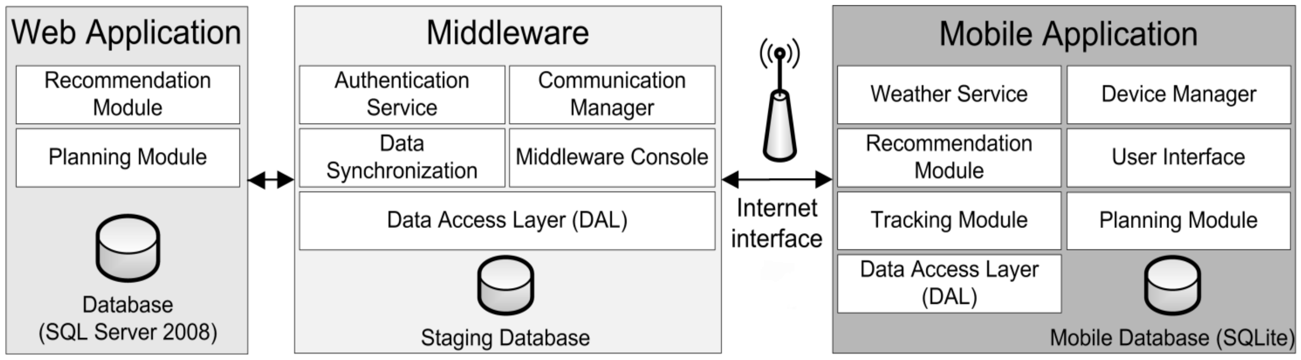


Figure 1 PSiS architecture overview

2.0 CASE STUDY CONTEXT

The necessity to discover which is the more adequate data transfer protocol and data serialization format to transfer information between a server and a mobile application came when we were developing a mobile application entitled PSiS Mobile.

This mobile application appears on the context of PSiS, which is a tour planning support web application that aims to define and adapt a visit plan combining, in a tour, the most adequate tourism products, namely interesting places to visit, attractions, restaurants and accommodations. This plan is generated according to the tourist's specific profile (which includes personal interests and values, wishes, constraints and disabilities) and the available transportation systems between the different locations [1].

In a first phase, tourists only interacted with PSiS through a web application accessible only from a browser, however nowadays it's indispensable to have a tool to assist tourists "on the field". Thus, we develop a mobile application entitled PSiS Mobile to integrate with PSiS.

PSiS is composed by three pieces (see figure 1), the server-side, the middleware and the mobile client. In the server there is a complete database with all the information about points of interest in a certain city/region and a complete user's portfolio, as well as, their travel history. The middleware was implemented to enable the communication between the server side and the mobile application.

The mobile client is a very important part of the system, because it establishes the bridge between the central services and the user visits. With a mobile device the user can see, on the go, the generated planning and the information about the nearby sights to visit, which are recommended according to his profile and current context. Trip planning can be rearranged according to current context, for example, if tourist is behind schedule the planning algorithm is executed to organize the original trip.

Since this application is an occasionally connected application (Smart client), a temporary database is used on the mobile device to enable access to parts of the data without being constantly consuming network traffic allowing the application to work without an internet connection (with some limitations, like no access to new points of interest).

After the user request a recommendation for a trip, all the necessary data is transferred from the server and stored on the mobile device. We have found this to be necessary, because of the mobile Internet low speed rates and its possible unavailability. The necessary data includes the information about all the points of interest that will be on the planning schedule, and other points of interest nearby the first ones. This approach is useful if the tourist wants to rearrange the original planning in real time.

What we pretend to do in this case study is to test the data exchange performance between the PSiS server application and the mobile one. To do this we have transferred points of interest data between the two sides. A representation of a point of interest data, structured as a XML, can be seen in figure 2.

Each point of interest is represented by 13 data fields where each one is formatted as a string field. The field which contains more data is the description, which can have more than 1000 characters. In average each point of interest has about 600 Bytes of data.

The tests were performed using 8 different Android mobile devices with different Android OS (Operating System) versions. We choose to use this broad range of devices and platforms to see if there are any significant differences between them or any evident changes in the results according to the hardware or software. These devices had different Chipset's, CPU's (from a single-core with 528 MHz to a dual-core 1.6 GHz), RAM quantities (from 288 Mb to 1GB), as well as, different versions of the Android Operating System, ranging from the 2.1 to the 4.1.2.

```
<PointOfInterest>
  <id>23</id>
  <name>Fantasporto - Porto International Film Festival</name>
  <description>Considered by the professional international magazine "Variety" as one of the 60 top film festivals of the world, and the best as far as the fantasy genre is considered, Fantasporto has become the most talked about film and culture event in Portugal. Organization: Cinema Novo CRL. Founding
member of the European Fantastic Film Festivals Federation and of the European Coordination of Festivals.</description>
  <poi_class_id>45</poi_class_id>
  <latitude>41.1537797</latitude>
  <longitude>-8.6210345</longitude>
  <address>Rua Aníbal Cunha, nº 84 - 4050 - 048 Porto</address>
  <phone>351 222 076 050</phone>
  <fax>351 222 076 059</fax>
  <email>pressfantasporto@mail.telepac.pt</email>
  <url>www.fantasporto.online.pt</url>
  <avg_cost>10</avg_cost>
  <avg_duration>180</avg_duration>
  <active>1</active>
</PointOfInterest>
```

Figure 2 Point of interest data represented as XML

Table 1 XML, JSON and protocol buffers structured data examples

XML	JSON	Protocol Buffers
<poi id="23">	{	poi {
<id>23</id>	"id":23,	id:23
<name>Fantasporto</name>	"name":"Fantasporto",	name:"Fantasporto"
<lat>41.1537797</lat>	"lat":"41.1537797",	lat:"41.1537797"
<lon>-8.6210345</lon>	"lon":"-8.6210345"	lon:"-8.6210345"
</poi>	}	}

The devices that we have used are listed below:

- HTC Hero, Desire and One S;
- Samsung Galaxy S and S2;
- Google Nexus S;
- BQ Edison;
- Huawei Ascend G300.

Four different sizes of data information were used to ensure more accurate results and evidence any major differences according to the quantity of data. The first dataset includes only the data from 1 point of interest, the second has information of 250 points of interest, the third has information of all the points of interest (461) present at PSiS database and finally we have used a heavier dataset with information of four times all the points of interest, which gives a total of 1884 points of interest.

Each test, that we have done, was performed 10 times per each mobile device and dataset. Between each test, the mobile device was turned off and the cache was clear. This is important to remove data in cache and to clean the device memory.

As expected the least powerful devices were slower than the best ones, and also there aren't any significant differences between the conclusions for them all. The results that we will present on this research paper are the average of ten runs, and using the Samsung Galaxy SII. We choose to leave out the information for the other mobile devices, since they followed the same pattern and did not bring any added value to the results analysis.

The only major difference that we have notice was related to the Android OS version, as we will see in the section 4.

■3.0 PROTOCOL EVALUATION

As is commonly known there are several ways to exchange information between a server and a client, but in this case we choose to study two of the most used data transfer protocols, the Java Socket API [2] and the HTTP with REST (REpresentational State Transfer) Web Services [3].

The other types of Web Services were left behind because they are too heavy for a mobile environment, *i.e.*, they have bigger headers than the REST architecture, thus increasing the amount of data traffic [4].

After select the data transfer protocols, we have selected the data structure formats to serialize the information. This is important in order to the two parts (server and client) "understand" each other. Since both protocols support the transfer of different file types, we choose to test three data structure formats – XML (eXtensible Markup Language), JSON (JavaScript Object Notation) and Protocol Buffers.

XML was chosen since it is one of the most popular data structure formats used to store information in a structured and hierarchical way. Also, it is widely used to store data to be exchanged between information systems.

Second is JSON [6] which has a structure identical to the XML, but tries to be a low-overhead format and nowadays it is being increasingly used.

Finally, we have the Protocol Buffers [7], which is a serialization format developed by Google Inc with the purpose to be lighter than XML, focusing on simplicity and performance. Protocol Buffers is very simple to use, because we only need to define how we want the data to be structured once. Then a generated source code, for that structure, is used to easily write and read the structured data to and from a variety of data streams.

It is implemented in Java, C++ and Python, so it can be used in Android but also in other mobile platforms. Also, it can also implement some data compression which improves the data file sizes. Since it has an open source license it is available to the public for free.

An example for each of these structure formats, applied to the information of a point of interest, can be seen on table 1. The file data sizes for each of these data structure formats is presented on table 2.

Based on this information we can confirm that JSON fulfills its propose, having a file almost 17% lighter compared to XML. However, Protocol Buffers is even about 16% lighter than JSON. So, the difference between this last two is almost the same difference between the XML and JSON. This represents that Protocol Buffers file is 30% lighter than XML.

These differences are also evident if we analyze table 1. In this example, the XML message has 95 characters, JSON 69 characters and Protocol Buffers message has 60 characters.

Raw socket was the first approach tested, since normally they are used to quickly exchange information [5]. First of all, a raw socket client and server modules were implemented.

Table 2 File sizes (in kB) for each data structure format and file number

	First	Second	Third	Fourth
XML	1.0	253	375	1875
JSON	0.779	227	313	1564
ProtocolBuffers	0.665	195	256	1276

Table 3 Transferred kB between the server to the mobile device

File Number	XML		JSON		Protocol Buffers	
	HTTP	Socket	HTTP	Socket	HTTP	Socket
First	1.34	1.28	1.23	1.07	0.851	0.799
Second	274.97	266.2	245.37	244.3	210.1	208.3
Third	405.33	389.9	337.28	337.3	275.4	272.4
Fourth	2000.2	1910	1700	1650	1400	1384

For each established connection, the server creates two threads: one for sending data to the client and another for receiving data from the client.

Since there are two different threads, one for sending and one for receiving data, the exchange can be performed asynchronously, avoiding waiting states on the client application.

With this protocol, message sizes were more compact since there aren't any headers (*e.g.*, HTTP or SOAP headers) -

table 3.

However, this system poses several problems to sockets management. Besides the apparent need to specify a hard-coded and very inflexible communication protocol, raw sockets also need further implementation for error detection and transaction control.

The other protocol that we test was HTTP, which is one of today's most popular client-server communication protocols. HTTP is a mature approach and a widely used protocol that already handle errors, simplifying its use and implementation. So, the errors that we have found using raw sockets don't emerged with this protocol.

The only downside, comparing to the raw socket communication protocol, is the size of the sent/received data frames. This mainly happens because of the HTTP header, which is added to the sent/received data.

The size of the header, on HTTP, along with the sent and received ACK packages to validate the transaction, varies between 6% and 10% of the size of the transferred data. For example, for a XML file with a size of 1.875 Mb, the client receives a total of 2.0 Mb (9% more than the original file size). The size of the headers on socket (which includes all the ACK) varies between 2% and 6%, meaning that it transfers less 4% data than HTTP.

However, the size of the data frames isn't the only metric that we use to compare both protocols. The download time is also very important when transfer information between two sides, since it influences the application responsiveness.

These protocols were tested using a normal Notebook PC working has a server and an IEEE802.11g wireless network to transfer the data between the two sides.

Analyzing table 4 we can see that raw socket protocol proved to be slower mainly because of the connection initialization, which is a time consuming process, especially when we try to control errors that may exist in the connection.

It was slower for the smaller files, however when the files were getting bigger, the results were better. This mainly happens because it needed to transfer less 100 kB than the HTTP protocol. The part that needs more time is the initialization part in order to control the errors and the creation of the socket's to transfer information between the two sides.

Another curiosity, from our tests, is that socket method proved to consume less system resources (CPU and memory) than the others because it doesn't have so many parsing routines. However, the whole process still takes more time to execute, which isn't so great for the system responsiveness.

Since users of this type of mobile applications (tourism) will use it more in an environment where no Wi-Fi connectivity exists, we test the download time difference between the Wi-Fi and the 3G network, both using HTTP protocol. The tested 3G network has a speed of 5Mbps and it had the signal at maximum level. As already stated in the literature [8] we also notice that the major problem of the 3G networks is the latency.

In the first file, which includes only the data from one point of interest, for all of the three file types (XML, JSON and Protocol Buffers) we can see that the download time is almost 1000% bigger than over Wi-Fi. As the file size is getting bigger the download duration difference is smaller. For the last file (the biggest one) the difference is around 600% more time. In practice this is the difference between waiting almost 3 seconds to download the file via Wi-Fi to wait over 18 seconds to download the file via 3G. To the user experience this is very relevant, and can compromise an application.

Based on this results we can conclude that sockets uses less bytes to transfer messages than HTTP, which can reduce mobile network costs since they can be expensive and we need to use the less bytes as possible. However, it is also important to consider the process duration in order to maximize the user experience. Taking into consideration this metric and the obtained results for all the file types and sizes, we can say that the HTTP protocol is better. Furthermore HTTP protocol is easier to use and implement, it already controls errors and is well established in the community, being used in a lot of server applications, making the migration to a mobile environment easier.

C onsidering these statements we choose HTTP protocol to exchange information between the PSiS server and PSiS Mobile application.

Table 4 Download duration for each protocol and data format serialization

File Number	XML		JSON		Protocol Buffers	
	HTTP	Socket	HTTP	Socket	HTTP	Socket
First	23.3	44.8	19.3	43.7	12	39.7
Second	757.6	808.1	665	702	420.8	475.8
Third	990.0	1010.6	740.4	755.6	564.3	570.1
Fourth	5176.3	5175.4	3631.8	3707.6	2314.3	2246.7

Table 5 Download time using IEEE 802.11G and 3G

File Number	XML		JSON		Protocol Buffers	
	Wi-Fi	**3G**	**Wi-Fi**	**3G**	**Wi-Fi**	**3G**
First	23.3	199.6	19.3	198.1	12	165.1
Second	757.6	5343.5	665	3422.8	420.8	3126.2
Third	990.0	6030.7	740.4	4691.5	564.3	3536.8
Fourth	5176.3	25323	3631.8	21628.9	2314.3	18084

■4.0 SERIALIZATION EVALUATION

After choose the exchange protocol we need to analyze which serialization format and respective parser is faster to use. To have a better understanding of the XML performance, we tested three different XML parsers:

- DOM (Document Object Model);
- SAX (Simple API for XML);
- XPP (XML Pull Parser).

DOM was chosen since it is the World Wide Web Consortium (W3C) standard and the other two because they claim to be the fastest XML files parsers.

The parsers that were used are the ones included in the Android OS, apart from the Protocol Buffers which is an external API. The packages that we use for each parse were the following:

- DOM – org.w3c.dom
- SAX – org.xml.sax
- XPP – org.xmlpull.v1
- JSON – org.json
- Protocol Buffers – com.google.protobuf

The first file, where we have used only the data of one point of interest was valuable to get a first look on the behavior of the mobile devices when few data bytes are parsed compared to bigger files.

In table 6, are described the parse times for each file and the correspondent parser. According to the results it appears that the fastest parser is Protocol Buffers. However, the second fastest depends on the Android OS version. From our previous publication [9] we can see that in Android 2.1 the second fastest was SAX Parser with XML. In these tests we can see that for Android 2.3.7 the seconds fastest is JSON and for Android 4.1.2 is the PULL Parser with XML.

This aroused our curiosity, since this was transversal for all the devices. The results only changed according to the Android version. So we analyze the Android source code and found that each parser (JSON for Android 2.3 and PULL Parser for Android 4.1) receive improvements in their implementations, turning them 20% to 30% faster than before. This was an excellent finding that has confirmed our results.

According to the results presented in table 6 we can verify that JSON is the fastest when parsing the first file. Also, the SAX Parser is similar to the Protocol Buffers, though the performance of the PULL Parser is very poor for small files, when compared to the results for bigger files.

Analyzing the second file, where the information about 250 points of interest were transferred, one of the most relevant findings are revealed. The XML parsing algorithms have significant performance differences. The DOM was the slowest and PULL proved now to be the fastest XML parser.

In the third file, which has the information about 461 points of interest, the results follow the same pattern, where Protocol Buffers was the fastest by a significant margin.

JSON behaved as expected, its serialization turns the file lighter than XML, but his decoder on Android 4.1 isn’t so good when compared with the PULL parser.

Finally, analyzing the more thorough test we can extract additional information from the obtained results. Remember that in this test we have used the information about 1884 points of interest. Comparing the third with the fourth file, we can observe that the processing time of DOM parser has been 6 times greater and the amount of data transferred is only 4 times the data transferred on the third test. This is mainly explained because of the limited mobile device memory. The operating system is always trying to get more and more memory and it slows down the entire process.

Notice that in some devices, the ones with less memory, DOM parser gave an “Out of Memory” error due to the mobile device lack of memory.

In this test we can denote a bigger difference in performance between Protocol Buffers and PULL or JSON. The difference of the parse duration between them passes from 30% to 40%.

What we can conclude from all of these results is that the parsing duration depends heavily on the parsing algorithm and there are significant differences between the Android OS versions which might be considered when implementing an application of this type. Only the extremes are equal for all the platforms, the lowest is DOM Parser and the fastest is Protocol Buffers.

S ince Protocol Buffers, which was the fastest, is an external library it not depends on the Android version, so the implementation was the same for all the tests.

Table 6 - Parse duration (seconds) for each format and Android version

File Number	SAX		DOM		PULL		JSON		Protocol Buffers	
	2.3.7	**4.1.2**	**2.3.7**	**4.1.2**	**2.3.7**	**4.1.2**	**2.3.7**	**4.1.2**	**2.3.7**	**4.1.2**
First	1.9	1.8	10.6	8.7	9.3	7.3	0.8	0.3	1.3	1.2
Second	219.7	211.7	640.9	615.6	205.7	161.1	186	186.8	119.9	115.2
Third	329.7	315.4	1120.8	1060.1	342	198.2	252.9	237	134.3	134.4
Fourth	1262.3	1200.6	6744.9	6514.3	1444.7	825.8	1175.2	887.4	551.9	503.1

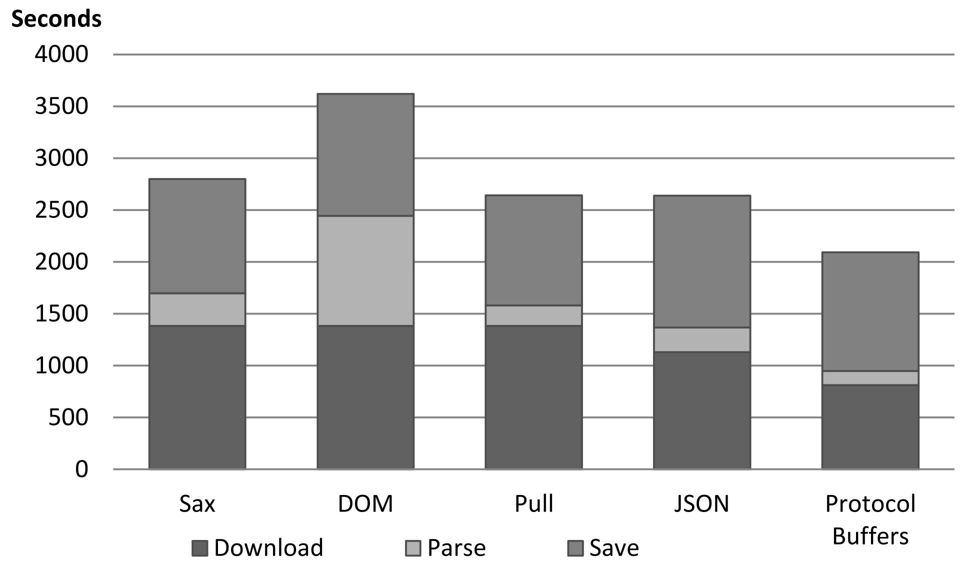


Figure 3 Data exchange process duration for 461 points of interest, for each parser, using HTTP protocol

The second fastest depends on the Android version, for Android 2.3 we have the JSON parser and for Android 4.1 we have PULL parser.

The CPU utilization data is very similar between all the parsers, the major concern are the 6 seconds that DOM puts the processor at 100%, which can represent a lot of battery spent.

With these results we choose the Protocol Buffers as the serialization method, since it not depends on the Android OS implementation and presented the best results. Also, it is available to use in other mobile platforms.

■5.0 CONCLUSION

The purpose of this study was to discover which technology/technique is more reliable and faster to use in order to transfer information between a server and an Android mobile application. Therefore, in this chapter we present our conclusions about the obtained results and what technique we choose to use and why we did it. Also, we present some considerations that we have learned and validate during these tests.

In theory, socket approach seems to be the right choice. However, in practice we have found some important disadvantages compared to other approaches, since it proved to be more error prone and slower. Analyzing the cost over benefit between this approach and HTTP, it was concluded that the socket gains on the transferred kB's between the two sides, don't outweigh the associated disadvantages. Also, raw sockets are much more complex and hard to work with. On the other hand, HTTP is reliable and is able to perform natively error handling.

After choose the transfer protocol we inspect the most commonly used data serialization formats to encapsulate our data to be sent over that protocol. Starting with XML, the case study revealed that after all it isn't so slow to parse, but instead it highly depends on the parser that we pretend to use. The biggest issue of this serialization format is the file sizes which are about 30% bigger than the messages created by Protocol Buffers. This happens because of the inclusion of multiple tags and it hasn't a data compression implementation. Then and as expected, since it is one of its claims, JSON files are smaller. However, depending on the Android version the native JSON parser can be slower than the best XML parser.

It is also interesting to analyze figure 3 where we present the overall process duration for each parser using the HTTP protocol to download the data. It is worth to notice that the average time to save the data in the database is almost 1.2 seconds. The parse time is the part of the all process that consumes less time. The most important is the download one.

Considering figure 3 we can see that Protocol Buffers is much faster than the others, and the difference between JSON, SAX and PULL parser isn't so significant.

According to the previous statements, the HTTP protocol in conjunction with Protocol Buffers was the mechanism that we choose to exchange information between our server and our mobile application, since it spent less system resources (therefore less battery) and less network data consumption. Thus, we minimize some of the limitations of mobile devices.

Another lesson that we have learned is that there is no advantage in sending fewer or huge information at once, but something in between them. If we send few information at once we have a great waste of time in the initialization of the communication, as we can see comparing the second and third files. However, if we send a lot of information at once, as done in the fourth file, we can experience some memory problems and thereby slow down the whole process. The best thing to do is to choose something in the middle, *i.e.*, medium-sized files. This happens because Android heap memory is limited to 16MB per application on the most available devices, and only the high-end ones have a limit of 24MB.

Another important note is to realize that 3G communications are very slow when compared to IEEE802.11 ones. The results established that the download duration difference can be 10 times slower for smaller files and 5 times slower to larger files, which represents a lot of time. when using a mobile network, it is necessary to optimize the applications in order to prevent overweight the network and to not decrease the user experience.

Finally, we have learned that it is worth investing some time in carrying out these small tests, because with this knowledge we can improve, a lot, the user experience. Has can be seen, for Android platform, the HTTP protocol and Protocol Buffers are so well implemented that it is worth to give a try, getting a fast and reliable solution to transfer information between a server and an Android mobile device.

Acknowledgement

The authors would like to acknowledge FCT, FEDER, POCTI, POSI, POCI and POSC for their support to GECAD unit, to the project PSIS (PTDC/TRA/72152/2006) and for the PhD grant (SFRH/BD/70248/2010).

References

[1] Anacleto, R., N. Luz and L. Figueiredo. 2010. Personalized Sightseeing Tours Support Using Mobile Devices. Human-Computer Interaction (eds. Forbrig, P., Paternó, F. and Mark Pejtersen, A.). IFIP Advances in Information and Communication Technology. Springer Boston. ISBN 978-3-642-15230-6. Volume 332/2010. 301–304.

[2] Harold, R. and M. Loukides. 2000. *Java Network Programming*. O'Reilly & Associates, Inc. Sebastopol. CA, USA.

[3] Fielding, R. and R. Taylor. 2002. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology (TOIT)*. 115–150.

[4] Pautasso, C., O. Zimmermann and F. Leymann 2008. Restful Web Services Vs. Big Web Services: Making the Right Architectural Decision. Proceedings of the 17th international conference on World Wide Web. 805–814.

[5] Pakin, S., V. Karamcheti and A. Chien. 1997. Fast messages: Efficient, Portable Communication for Workstation Clusters and MPPs. *Concurrency IEEE*. 5(2): 60–72.

[6] Crockford, D. 2006. JSON: the Fat-free Alternative to XML. In Proc. of XML.

[7] Google Inc. 2013. Protocol Buffer. http://code.google.com/apis/protocolbuffers/docs/overview.htm.

[8] Inamura, H., G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov. 2003. TCP Over Second (2.5 G) and Third (3G) Generation Wireless Networks. RFC3481.

[9] Anacleto, A., L. Figueiredo, A. Almeida and P. Novais. 2013. Server to Mobile Device Communication: A Case Study. Ambient Intelligence - Software and Applications (eds.) Berlo, A., Hallenborg, K., Rodríguez, J. M. Corchado, Tapia, I. and Novais, P. Springer - Series Advances in Intelligent Systems and Computing. ISBN 978-3-319-00565-2. 219: 79–86.