Cloud session maintenance to synchronise HbbTV applications and home network devices

Mikel Zorrilla, Iñigo Tamayo, Angel Martin and Igor G. Olaizola

***Abstract*— Second screen services encourage TV audience to enjoy new forms of interaction engaging users around TV content as a main thread. This paper describes a standard-based solution for second screen services synchronised with the broadcast content. The user perceives an enhanced broadcast experience enriched with multimedia, textual and social Internet content through multiple devices. The presented end to end solution delegates to a server the cloud session maintenance in order to pair and synchronise HbbTV applications and HTML5-based second screen ones overcoming existing heterogeneous network interfaces barriers of current technological alternatives. The server decides dynamically the behaviour of the different applications regarding the user context, according to his preferences, device features and number of simultaneous views. It also manages the user interaction providing a full synchronised experience thanks to an event-driven mechanism on top of Websockets and AJAX. The paper analyses the performance of the proposed system evaluating the user interaction latency, the concurrency volume of the server and the interdependence, concluding this solution as a suitable approach for broadcast-related second screen services.**

***Index Terms*— Multimedia systems and services, DTV and broadband multimedia systems, Multimedia devices, Set-top box and home networking, HbbTV, HTML5, smart devices, broadcast interactivity.**

I. Introduction

USER experience around television has changed a lot. Second screen paradigm is increasing daily. Many households are already multi-display, using their smartphone, tablet or laptop for sharing with friends through social networks and to interact directly with the programme reaching metadata or additional content. The main tasks that users perform on second screen services include discovery of related contents, finding supplementary information, participation in live shows, shopping of promoted items, and social sharing, faced by different stakeholder actors.

On the one hand, Smart TVs, which have become a commercial success worldwide, provide OS-based heterogeneous platforms to access applications and services through markets or stores. We can find manufacturer created vertical platforms as Samsung Smart TV or LG Smart TV, or proprietary horizontal approaches as Google TV or Apple TV. These OS-based applications are ready to reach Internet services through the TV but they are not linked to the mainstream, the broadcast content. Users have a discontinuous experience with these platforms, using the device to watch broadcast content or to access applications, ignoring profits of multiple synchronised experiences to capture audience interested in finding further information about TV shows they are watching [1].

On the other hand, there are standard approaches involving broadcasters to achieve a synchronised experience exploiting metadata and additional content trough the TV, related to the broadcast content. HbbTV [2] [3] is a pan-european specification published by ETSI[1] to allow broadcasters application signalling and delivery through the carrousel, rendering the application with a CE-HTML based browser in the TV or set-top box bringing a new unexplored market.

Unfortunately, neither HbbTV v.1.1 nor v.1.5 specifications deal with the migration of services to other home devices, such as tablets and smartphones, to achieve a multi-screen user experience synchronised with the broadcast content. These features will hopefully be included in the future HbbTV v.2.0 specification but a solution is needed to overcome this limitation for current deployments.

In this paper we present an end to end solution to synchronise HbbTV applications (both v.1.1 and v.1.5) with mobile devices (e.g. Android or iOS devices) through standard HTML5 applications, using cloud session maintenance in the server side. Thereby the server keeps and manages session variables to identify the different users and their devices, pair them dynamically and adapt the service to the user context, delivering the different target contents to the specific devices.

Section 2 presents the cloud session maintenance architecture approach. Section 3 describes performed validation experiments. Section 4 details the obtained results for the most critical features in terms of low-latency synchronisation regarding the concurrency of the server. Finally, section 5 shows the conclusions.

Manuscript received May 1, 2013.

M. Zorrilla is with the Department of Digital Tv & Multimedia Services, Vicomtech-IK4 GraphicsMedia.net, Donostia-San Sebastián, 20009 Spain (phone: +34-943-309-230; fax: +34-943-309-393; e-mail: mzorrilla@vicomtech.org).

I. Tamayo is with the Department of Digital Tv & Multimedia Services, Vicomtech-IK4 GraphicsMedia.net, Donostia-San Sebastián, 20009 Spain (e-mail: itamayo@vicomtech.org).

A. Martin is with the Department of Digital Tv & Multimedia Services, Vicomtech-IK4 GraphicsMedia.net, Donostia-San Sebastián, 20009 Spain (e-mail: amartin@vicomtech.org).

I. Olaizola is with the Department of Digital Tv & Multimedia Services, Vicomtech-IK4 GraphicsMedia.net, Donostia-San Sebastián, 20009 Spain (e-mail: iolaizola@vicomtech.org).

[1] Last version: ETSI TS 102 796 v1.2.1 in November 2012

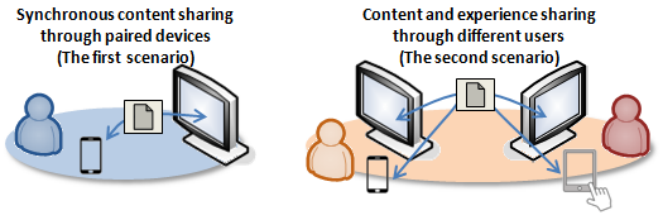

Fig. 1. Scenario diagrams

II. Cloud Session Maintenance Architecture

The different scenarios that compose the scope of the solutions meet additional contents or information around the broadcasted content empowered by means of an HbbTV application on the TV and synchronised HTML5-based applications on the second screen devices (see Fig. 1). The first scenario involves the synchronous content sharing from the both paired devices on a living room (TV and second screen device), while the second scenario faces contents sharing from any of the involved devices, to friends' TV or mobile through their also paired and synchronised applications even if they are geographically distributed. These scenarios enable the broadcasters to implement Customer Relationship Management (CRM) services with the desire to improve their services and retain their customers [4] [5].

For example, a user, who has already paired a tablet to his Smart TV, is watching a football match on the TV set with his family while he accesses additional contents through the paired tablet application, such as other sport results. Suddenly, a new scored goal of other match is played on the tablet and he wants to share it with other households overlaying the goal replay over the broadcast signal. He can perform this action on the tablet and send it to the TV. The server receives from the tablet the "send it to the TV" event and generates a command for that TV's application in order to show that video (the first scenario). Besides, he shows this goal to a friend of him, which is in another country watching the same match and having the same synchronised experience through a smartphone and a TV (the second scenario).

To address the goals specified in these scenarios our approach defines an architecture based on three main modules (see Fig. 2): 1) A Node.js[2] web server to manage the cloud session maintenance and the behaviour of the applications, 2) HbbTV applications for the Smart TVs and 3) HTML-based applications for the second screen devices (tablets, smartphones, laptops, etc.).

Node.js is framework for developing high-performance concurrent programs that do not rely on the mainstream multithreading approach but use asynchronous I/O with an event-driven programming model [6]. This enables the server to create low coupling bi-directional communication channels between client applications. It receives user interaction events and generates reaction commands to push unleashed events on other applications.

[2] http://nodejs.org

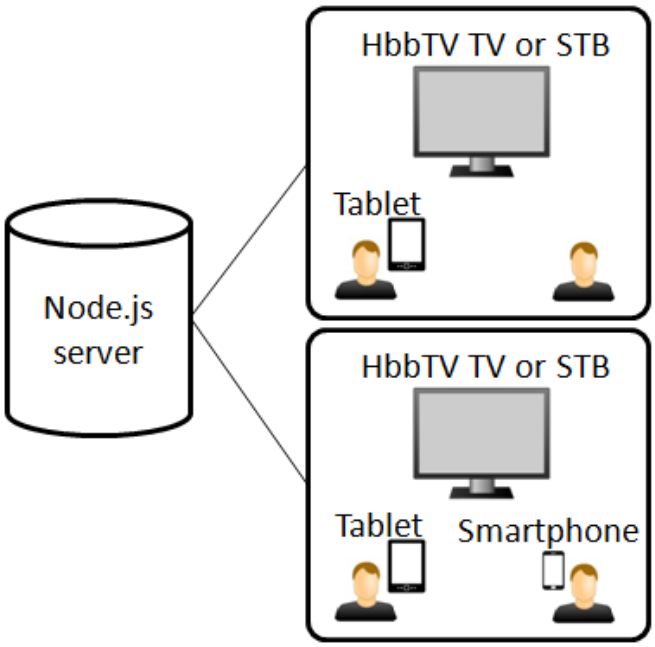

Fig. 2. The different modules of the architecture

A. System Workflow

The service always starts with an HbbTV application associated to a broadcast content. Each TV application is linked to a unique *session vector* stored in the server which contains contextual information: a) a unique session token, b) information about already synchronised mobile devices and c) user preferences information.

The TV application is not paired with any devices by default, but the user will be able to pair different mobile devices via a web browser. The TV application shows a QR code which contains a URL to the server, including REST-based information to identify that mobile device with the TV (e.g. the unique session token).

From that moment on, the *session vector* will contain information about the second screen paired with the TV and so the TV application could change it behaviour to adapt to that new context. For example, the HbbTV application shows a menu with some options (one of them is to pair the TV applications with a second screen device) while the user is watching a live event broadcast content. Once a tablet is paired, the TV menu disappears going back to full-screen broadcast signal, meanwhile a secondary display menu is available in the tablet enabling sharing actions of application contents to the TV.

Additionally, the service could be configurable by the user, defining his preferences of what kind of information wants to see in each device, how long overlay it in the TV, etc. All this information will be stored in the *session vector* to influence the parameters of the commands that the server generates.

B. Server components

Going into detail, the server has different components to deal with the requirements of the service (see Fig. 3). On the one hand, it has an HTTP server for the delivery of the HTML-based application to the end-clients; this includes the HbbTV application to the Smart TV and the HTML5 applications to the second screen devices. On the other hand, it

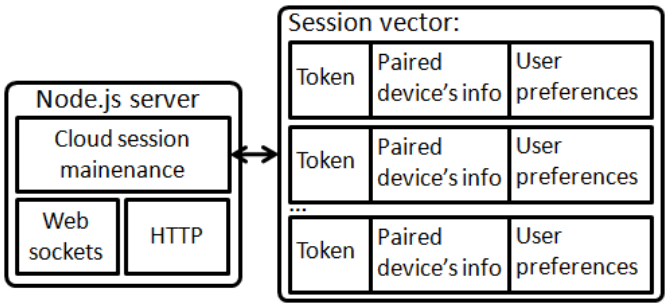

Fig. 3. The components of the server and the *session vector* structure

has a Websocket module to establish a bi-directional communication channel between the server and the end-devices. This channel is used to receive the user interaction events by the server and to delivery reaction commands to the other synchronised applications. Finally, the server has a *cloud session maintenance* module which takes charge of: maintaining and updating a *session vector* for each HbbTV application; creating the commands and events depending on the multi-device behaviour of the service; and managing the Websocket module and communications.

Concerning latency and data traffic overhead Websockets are the best and more efficient approach for the event communication between the Web server and the clients. Nevertheless, there are some technology restrictions to use them. HbbTV v.1.1 does not have support for Websockets but it is already included in version v.1.5. Regarding the web browsers of the mobile devices, Websockets are included in the roadmap of all of them, but implementations are still missing in some cases. While iOS Safari and Blackberry browser support Websockets, latest versions of Android Browser and Opera Mini do not[3]. The system gets over this limitation using polling mechanisms in the client with AJAX to detect remote activity in the server.

End client HTML-based applications, both TV and second screen device applications, are built over conditional aspects with different behaviour according to: a) presence of second screen devices paired with the TV, b) personal configuration of the user preferences and c) capabilities of the devices in terms of Websockets support, screen size, etc.

C. Main advantages

The architecture proposed in this paper brings some significant advantages from other solutions to pair second screen devices to the TV. Most of them are based on local network communication protocols such as UPnP or DLNA. They require configuring the same Internet access point for the household devices. The cloud session maintenance solution pushes to the server the device pairing management. This way it allows the communication between devices connected to Internet no matter how. For example, the user can pair a TV connected to the broadband home Internet connection with a smartphone connected via 3G to Internet. This feature enables multi-user experiences, with the synchronisation and interaction among two people which are not physically

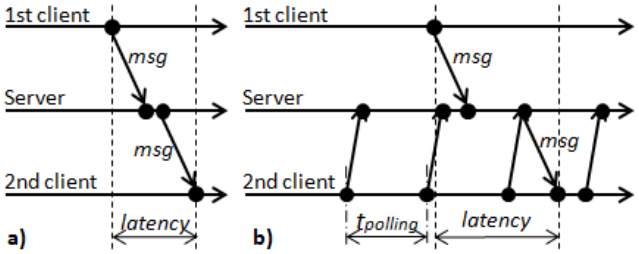

Fig. 4. Latency measure using a) Websockets and b) AJAX

together. Finally, it does not require local communication capabilities to the HbbTV devices (UPnP, DLNA, etc.).

D. Risks

Nevertheless this scenario has some risks that should be considered, mainly in terms of latency, concurrency and security. Local network based pairing solutions offer low latency characteristics and secure protocols are not mandatory because devices need to be connected to the Home Network (same Internet access point). But cloud-based solutions must keep latency performance in order to grant an appropriate quality of experience to the user even under stress conditions of the platform with massive users performing concurrent requests to the system [7] [8]. Regarding security, this aspect is not afforded in this paper because it can be faced by any solution on top of the proposed architecture [9]. For example, [10] presents the challenges of building web-based ubiquitous applications over the Webinos[4] platform and facing the privacy and security aspects. By contrast, this paper shows a detailed latency and concurrency analysis of the Node.js-based server for cloud session maintenance services to synchronise HbbTV applications and home network devices as a second screen.

III. VALIDATION EXPERIMENTS

In order to measure the latency of the architecture proposed in this paper, two HTML-based applications have been run in a laptop, connected through a local network to the Node.js server. One of the applications acts as the first client, sending the timestamp of the system as a message to the server. The other application is the second client, waiting for a message from the server with the timestamp of the first client to measure the round trip time. This means the second client compares the system timestamp in the moment it received the message and the timestamp received in the message (both applications are in the same laptop so there are not clock synchronising problems). This result is the latency parameter since a user sends an event from one device, until the other device gets awareness, e.g. there are two households in front of a TV, one of them watching interesting additional information related to the content in the TV. Pressing a "Show in the TV" button he just pushes that information to the TV so users can watch it overlaid into the broadcast content.

The values depicted below have been obtained for two scenarios: using Websockets and using AJAX, for those

[3] March 2013, StatCounter GlobalStats http://caniuse.com/#feat=websockets

[4] Webinos is an EU FP7 ICT funded project aiming to deliver a platform for web applications across mobile, http://www.webinos.org/

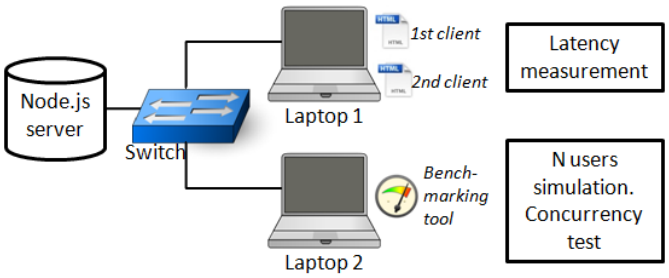

Fig. 5. Architectural diagram for the validation experiments

devices which do not support Websockets in their web browsers.

In the first case, the HTML-based applications open a bi-directional communication channel with the server through Websockets. This way, the first client sends the timestamp to the server, and the server forwards it in a best effort way to the second client (see Fig.4a). WebSockets are optimised to reduce communication cost over the Web to a minimum, with a header of only 2 byte compared to the 8 Kbyte header limit for most HTTP messages [11] [12].

In the other scenario, the first client opens an AJAX communication to send the timestamp to the server and the second application is recurrently asking through AJAX for a new event to the server. For the experiments, in order to ensure smooth usability with a high Quality of Experience (QoE) we have defined a 100 ms polling interval in the second client to ask for changes in the server. This has a direct impact in latency, adding an extra delay which has to be considered in terms of low-latency needs, and the network load (see Fig.4b) [13].

In order to obtain results closer to a public service deployment, benchmarking tools have been employed to simulate massive concurrent requests. This way we assess how its volume affects to the server in terms of latency (see Fig.5). We want to establish how many simultaneous users can be managed by the server keeping a minimum value of latency to provide a suitable QoE to the user.

For the simulation of multiple simultaneous AJAX connections we used JMeter[5]. Apache JMeter™ is open source software designed to test the functional behaviour and measure performance of web applications.

Due to unavailability of a benchmarking tool, we have developed a test environment to simulate multiple and simultaneous connection for Websocket technology on top of Socket.IO-benchmark[6] and Socket.IO[7] framework, hence we build and adapt them to be used with the Node.js-based server.

Moreover, the same experiments have been done using a server harnessing a variable number of CPU cores to profile the weight of the processing capabilities of the server on the performance. The Node.js-based server has been executed over an x86 64 bit PC with four CPU cores of 2.4 GHz where the OS is Debian 6.0 64bits.

Section 4 shows the obtained values of the latency regarding the number of concurrent connections to the server

[5] http://jmeter.apache.org/
[6] https://github.com/michetti/socket.io-benchmark
[7] http://socket.io/

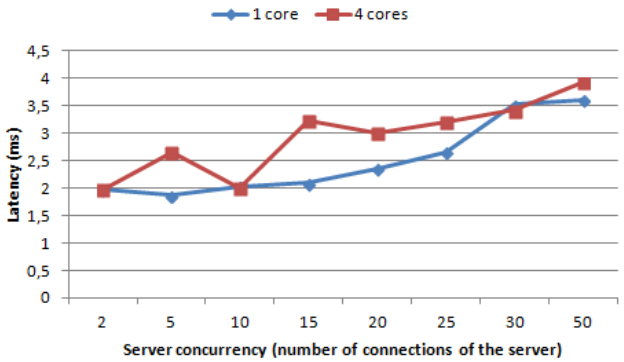

Fig. 6. Latency of the system with a low number of concurrent connections using Websockets

and the performance of the server in terms of hardware resources. Moreover, the behaviour of the different communication protocols for these parameters has been analysed.

IV. Results

This section details the obtained latency results using the test suite and environment described previously. Using Websockets in a local network, the proposed architecture achieves a latency score of 1,97 ms since the user presses a button on one device, until he notices the reaction of the service in the other device. These tests have been done using a single CPU core on the server and a four-core CPU. If we analyse the performance for a variable number of users, a single user using two devices obtain the same latency results mentioned before. For a low number of concurrent connections with the server the latency values are from 1,9 to 4 ms. In this cases, the OS multi-core management of the server increases slightly the latency (see Fig 6.).

Testing how the latency behaves when increasing the number of devices simultaneously sending events to the server, the CPU of the server is the bottleneck. While there is a *stable zone* with an appropriate latency threshold with a server of one and four cores, increasing the number of concurrent connections both react different. Once the CPU is overloaded, the latency increases exponentially. Fig 7. shows how the *stable zone* still continuous for a four core server for 3000 simultaneous connections while the one core server is drastically overloaded.

Same tests have been performed with AJAX asynchronous connections between the clients and the server for those clients which do not implement Websockets. As seen in Fig. 4 the client architecture to notice activity in the server needs recurring connections from the second client to the server. The tests have been done with a 100 ms slot from one connection to the other so it adds 50 ms of latency of average since all the latency values have been calculated with the average of 100 measures.

For better latency values we could reduce the 100 ms slot but a trade-off is needed between the latency and the overhead of the network and server in terms of applications design. Fig. 8 shows the latency using AJAX for a low number of

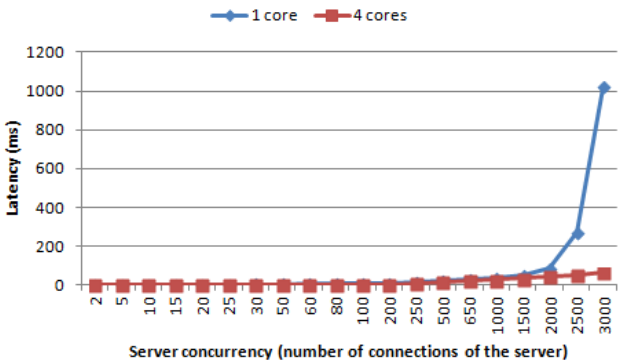

Fig. 7. Latency of the system with a high number of concurrent connections using Websockets

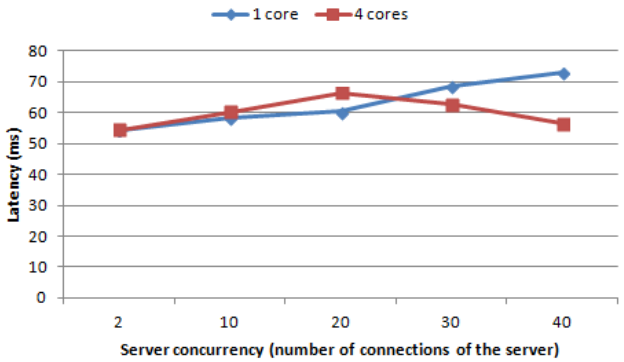

Fig. 8. Latency of the system with a low number of concurrent connections using AJAX

simultaneous connections to the server with a one-core and a four-core server.

The minimum latency value obtained is 54,47 ms, which matches 4,47 ms to the system architecture since 50 ms are added in the client applications design. Comparing Websockets and AJAX, there is an increase of 226% using AJAX, because it needs to open the communication channel each time. Using Websockets the applications open the socket session at the beginning and then just send the messages. But even with AJAX, the system latency is very suitable for a wide range of applications under these conditions.

Using AJAX has a higher impact in the server CPU so it accepts less simultaneous connections inside the *stable zone* latency. Fig. 9 shows the latency for a number of AJAX connections from 2 to 1300.

V. Conclusions

This paper introduces a cloud session maintenance architecture to synchronise HTML-based connected TV applications (HbbTV) and home network devices such as smartphone or tablets via their web browsers and provide concurrent second screen services.

Instead of turning on local network communication protocols to pair the second screen devices with the TV set, we define a server-based architecture that pushes to the cloud all the pairing responsibilities. It removes local area network constraints for the home multimedia devices such as using local network communication protocols (UPnP, DLNA, etc.) and share the same access point for household devices, and extends user synchronised experiences to Internet connected applications.

Moreover, different communication protocols have been analysed to send interaction events between the different clients and the server. On the one hand, we employ Websockets as the most efficient bi-directional communication channel. But for some devices, such as HbbTV v.1.1 equipment, some smartphones and tablets that do not support Websockets, an alternative architecture is proposed on top of AJAX.

The validation experiments have been described to test the proposed architecture in terms of latency and to parameterise the server performance for a high volume of users accessing concurrently. All the experiments have been carried out over a Node.js server with a variable number of CPU cores and connecting all the devices through a local network.

The results of the validation experiments conclude that the latency of the proposed architecture is suitable for most of the second screen services that can be envisaged over this solution. Websockets provide a much better latency mainly related to the provided bi-directional communication channel capability enabling the server to send events asynchronously to the clients. Nevertheless, an AJAX-based solution could be also adequate for a wide range of less demanding second screen applications.

Incrementing the number of simultaneous connections that are exchanging information with the server, there is a *stable zone* where the latency value remains quality threshold. This shows that the server architecture is ready to perform a multi-device ecosystem of many users interacting with synchronised contents concurrently. Nevertheless, there is an inflexion point where more users accessing simultaneously affect drastically to latency. This point is closely related to two main factors: 1) the communication protocol used between the clients and the server, being Websockets the most efficient solution dealing with a higher number of simultaneous connections without a negative impact on the latency and 2) the available CPU cores on the Node.js-based server. Using a single core CPU server the inflexion point rises with a limited number of concurrent connections while with four cores the number of connections accepted on the *stable zone* increase. These results demonstrate that the architecture is suitable for a high number of users using simultaneously the service. So, in order to manage adequately to a volume of concurrent requests for the latency *stable zone*, the server just must dimension CPU available hardware.

To conclude this paper provides a suitable end to end solution based on a cloud session maintenance architecture that allows synchronised second screen services with Connected TVs and mobile devices (smartphones and tablets). This architecture performs a solution over already commercialised devices such as HbbTV compatible Connected TVs (v. 1.1 or v.1.5) and Android, iOS, Windows, Blackberry, etc. smartphones and tablets with a standard mobile web browser.

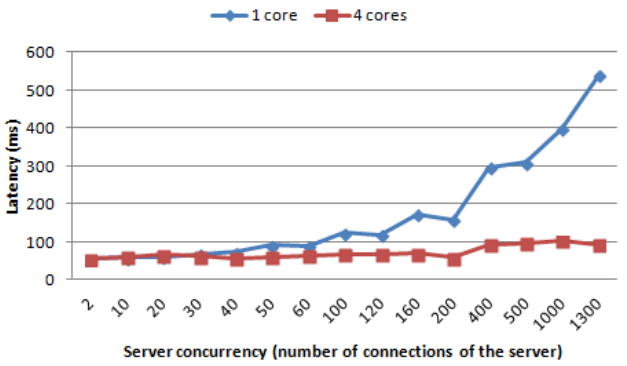

Fig. 9. Latency of the system with a high number of concurrent connections using AJAX

REFERENCES

[1] Lochrie, M. & Coulton, P. Mobile phones as second screen for TV, enabling inter-audience interaction. In Advances in Computer Entertainment Technology (ACE '11), ACM, New York, USA

[2] Merkel, K. HbbTV - a hybrid broadcast-broadband system for the living room. Published by the European Broadcasting Union, Geneva, Switzerland, Q1 2010. ISSN: 1609-1469.

[3] Merkel, K., "Hybrid broadcast broadband TV, the new way to a comprehensive TV experience," Electronic Media Technology (CEMT), 2011 14th ITG Conference on , vol., no., pp.1,4, 23-24 March 2011

[4] De Sutter, R., & Nachtergaele, L. Realizing CRM in the broadcast industry by using the second screen.

[5] De Sutter, R., Matton, M., Laukens, N., Van Rijsselbergen, D., & Van de Walle, R. (2011). MediaCRM: enabling customer relationship management in the broadcast. In Web Information Systems and Mining (pp. 19-26). Springer Berlin Heidelberg.

[6] Tilkov, S. Vinoski, S Node.js: Using JavaScript to Build High-Performance Network Programs. In: Internet Computing, IEEE 2010. Date of Publication: Nov.-Dec. 2010. Volume: 14 , Issue: 6. Page(s): 80 - 83

[7] Andrew Turner, Andrew Fox, John Payne, Hyong S. Kim, "C-MART: Benchmarking the Cloud," IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 6, pp. 1256-1266, June, 2013

[8] Zibin Zheng, Xinmiao Wu, Yilei Zhang, Michael R. Lyu, Jianmin Wang, "QoS Ranking Prediction for Cloud Services," IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 6, pp. 1213-1222, June, 2013

[9] Stephen Farrell, "Leaky or Guessable Session Identifiers," IEEE Internet Computing, vol. 15, no. 1, pp. 88-91, Jan.-Feb. 2011

[10] Desruelle, H., Lyle, J., Isenberg, S., & Gielen, F. (2012). On the challenges of building a web-based ubiquitous application platform.

[11] Gutwin, C.A., Lippold, M. and Graham, T.C. Real-time groupware in the browser: testing the performance of web-based networking. In Proc. CSCS 2011, ACM Press (2011), 167-176.

[12] Lubbers, P., & Greco, F. (2010). HTML5 web sockets: A quantum leap in scalability for the web. SOA World Magazine.

[13] Moberg, P. (2013). Event-driven interactivity in application-based TV-programs (Doctoral dissertation, Uppsala University).

M. Zorrilla is with the Department of Digital Tv & Multimedia Services, Vicomtech-IK4 GraphicsMedia.net. He received his Telecommunication Engineering degree in 2007 from Mondragon Unibertsitatea (Spain) and an advanced degree in Computer Science from UPV-EHU University of the Basque Country (2011-2013, Spain).

He focuses on the research lines related to multimedia services and interactive media technologies. During his studies he worked in the area of communications of IK4 Ikerlan Research Centre (www.ikerlan.es) (2002-2006). Afterwards, he developed there the End of Degree Project about The Transport of Multimedia Traffic With an "Industrial Ethernet" Communication Bus (2006-2007). Since 2007 he is working at Vicomtech-IK4, where he designs, develops and leads projects of the Digital Television and Multimedia Services area.

I. Tamayo is with the Department of Digital Tv & Multimedia Services, Vicomtech-IK4 GraphicsMedia.net. He received his Computer Science Engineering degree in 2007 from Mondragon Unibertsitatea (Spain).

He focuses on the research lines related to multimedia services and interactive media technologies. He developed there the End of Degree Project developing a Quality Management System on Fundación Legarra Etxebeste. In 2007-2008 he collaborated with Geoaim in the development of a Geographical System Information. Since 2008 he is working at Vicomtech-IK4, where he designs and develops projects of the Digital Television and Multimedia Services area.

A. Martin is with the Department of Digital Tv & Multimedia Services, Vicomtech-IK4 GraphicsMedia.net. He received his engineering degree in 2003 from University Carlos III, Spain.

He starts his Phd in the Communications and Signal Processing Department of the University Carlos III in the video coding research area in 2003. At the same time he collaborates with Prodys developing a standard MPEG-4 AVC/H.264 codec for DSP. In 2005 he starts to work on Telefonica I+D in projects related to multimodal interactive services for Home Networks. Also in Telefonica I+D, he goes deeper into image processing area in terms of 3D video and multiview coding. From 2008 to 2010 he worked in Innovalia as R&D Project consultant related with smart environments and ubiquitous and pervasive computing. Currently he is on Vicomtech-IK4 managing and developing R&D projects around multimedia content services.

I. Olaizola is the head of Digital TV and Multimedia Services Department in Vicomtech-IK4 GrpahicsMedia.net, Spain. He received his Engineering degree from University of Navarra, Spain (2001).

He developed his Master thesis at Fraunhofer Institut für Integrierte Schaltungen (IIS), Erlangen -Germany- 2001 and currently he is preparing his PhD in Computing Science and Artificial Intelligence at University of Basque Country. He has participated in many industrial projects related with Digital TV as well as several European research projects in the area of audiovisual content management. His current research interests include multimedia content analysis frameworks and techniques to decrease the semantic gap.