International Journal of Web Engineering 2013, 2(1): 1-8
DOI: 10.5923/j.web.20130201.01

Architecture for Real Time Communications over the Web

S. Panagiotakis[1], K. Kapetanakis[2,*], A. G. Malamos[2]

[1]Department of Sciences/Division of Computer Science, Technological Educational Institute of Crete, Heraklion, Crete, GR 71410, Greece
[2]Department of Applied Informatics and Multimedia, Technological Educational Institute of Crete, Heraklion, Crete, GR 71410, Greece

Abstract The emergence of HTML5 and other associated web technologies can shape a diversity of future applications, where the client-server operations will be obsolete. In particular, the Media Capture and Streams API of HTML5 enables third party access to multimedia streams from local devices. Enriched with a WebSockets implementation, a web application can communicate, stream and transfer media or other data to its clients at real time to support a full collaborative environment. In this paper, we introduce an architecture that capitalizes on the above technologies to enable real time communications over the web. We also demonstrate the web applications we have developed in this context for live video streaming and web video chat with no requirement for any plug-in installation.

Keywords WebSockets, HTML5, Video, Streaming, Conference, Web, Real-time Communications, Get User Media, Web RTC

1. Introduction

So far, real time media communication between various client devices, either one way (streaming) or two (chat or conference), was, more or less, a static and monolithic operation dominated by several platform-specific solutions. In particular, the streaming of media required the setup of dedicated streaming servers, the installation of the appropriate standalone applications at client side and, obviously, the support of the corresponding streaming protocols for transferring the streamed packets. As far as it concerns the latters, a whole family of private and standardized ones is provided. Similar is the view with respect to chatting and conferencing, which additionally require the mediation of a session manager between clients and the support of the corresponding session protocols. With respect to communicatingat real time via the web, until recently the streaming of media over HTTP was just a myth, while the receipt of streaming media via web could be accomplished only with the installation of the appropriate third party software (browser plug-in) to receive and process the data streamed from the server. Additionally, the popular media players provide plug-ins for most browsers to allow video and audio streams to be played back over the web. Web chatting and conferencing is also possible only via plug-ins. SIP (Session Initiation Protocol) [1] and XMPP (eXtensible Messaging and Presence Protocol)[2] are the most popular protocols for such uses.

However, the emergence of HTML5[3] and other associated web technologies have drastically changed the whole view to a dynamic, browser-friendly and platform independent approach. This is due to the fact that HTML5 introduced several extended functionalities to web-browsers changing the way data are transferred, visualizations are displayed and graphics are processed using hardware acceleration. This is mostly accomplished via several JavaScript libraries and custom JavaScript programming which allow to web-pages to gain access to various device features provided for media access and customization. In that context, the installation of flash player is not mandatory for video streaming any more, since HTML5 provides an element with the tag name “video” that can substitute the requirement for any such plug-in. Furthermore, images can be loaded in an element with the tag name “canvas”, a container which can be used to draw graphics on the fly with JavaScript. The canvas element is supported anymore by the most popular desktop and mobile browsers. Technologies such as WebGL(Web Graphics Library)[4], SVG (Scalable Vector Graphics)[5] and Quartz 2D[6] can be combined with canvas element to draw 2D and 3D graphics with support for user interaction. More critical, the Media Capture and Streams API[7], part of the general Device APIs[8], enables access via the web to a user’s microphone and camera device. To this end the GetUserMedia method is defined. Hence, the live streaming of media, audio and video, from a user over the web can be now a reality.

* Corresponding author:
kapekost@epp.teicrete.gr (K. Kapetanakis)
Published online at http://journal.sapub.org/web

Additionally, an innovative approach takes place in the recent browser releases with respect to the data transferring protocols over TCP/IP. WebSockets[9][10] use standard HTTP signaling to establish a persistent bidirectional TCP connection and transfer data, in the form of Web Sockets frames, between web clients and web servers. With WebSockets web push, that is pushing data from a web server to its clients, can be a reality. In many web servers, WebSockets are ready and tested, following the introduction of the new HTML5 APIs. WebSockets Servers can be found written in many programming languages including C, Python and Java.

In the same context, WebRTC[11] is an open project supported by Google, Mozilla and Opera that aims to bring high quality Web Real Time Communications between browsers using simple JavaScript and HTML5. WebRTC uses the aforementioned getUserMedia method to access a peer's microphone and camera and stream media data to another peer browser and vice versa. The peer-to-peer connections are established via the Peer-to-Peer Data API which implements JSEP (JavaScript Session Establishment Protocol)[12]. JSEP signaling is used in the WebRTC framework as SDP in SIP-based communications that is to describe and negotiate a session between two browsers with the mediating session manager. WebRTC does not mandate the use of any session protocol, such as SIP or XMPP, enabling implementation differentiation. Ericsson labs in[13] presented the first implementation of WebRTC in 2011.

In this paper, we introduce an architecture for real time communications over the web. Using various HTML5 features and APIs we have created cross-device web applications that can access the web-camera on client side, capture video and transmit it over the web using a WebSocket connection with an associated server. In this paper we present the generic architecture that enables such communications and demonstrate the two web applications we have developed for real time communication over the web. The first one enables live video streaming over the web and the second the video chatting between two browser clients. Both solutions do not require the installation of any plug-in or the support of any session description protocol and have been successfully tested on several devices, mobile or desktop. The only requirement is for a browser compatible with HTML5 and WebSockets. So far, we are not aware of other similar implementations in scientific literacy apart from the sophisticated WebRTC. The rest of the paper is organized as follows: section 2 presents some related work, section 3 introduces to the proposed generic architecture for real time communications over the web and section 4 demonstrates the two web applications we have developed for live video streaming and web video chatting. Finally section 5 concludes the paper.

2. Related Work

HTML5 is rapidly implemented in the most popular browsers. Additionally, portable devices such as tablets, SmartPhones, even televisions are equipped with web - browsers. The implementation of HTML5 in these devices has already started. Not far from reality, smart TVs are already launched with HTML5 enabled browsers and are able to stream video for broadcasting television programs [14]. Thus, each application totally compliant with HTML5 will be device independent. However in each device the input facilities vary. HTML5 has already taken into account such occasions providing special APIs for touch events, orientation alteration and other input data from new sensors[8].

Researchers have recently rushed into video implementations on web applications using HTML5. In[15] Daoustet. al. present the HTML5 video element and demonstrate an implementation of CSS that transforms the displayed video on the fly. Additionally in this paper, they analyze video streaming over HTTP and discuss the ability of peer to peer streaming. HTML5 has also been used in real-time communications. For example, a web application for presentations is introduced in[16]. In this paper, the authors propose a framework for displaying slides synchronized with a video stream over the web. To this end, some JavaScript functions update the current slide according to the timing in a SMIL document.

With respect to real time conference-like communications the SIP is the dominant signaling protocol so far. In such communications a Multi-point Control Unit (MCU) is implemented to control the flow of data among the connected users in the optimal way. Han et. al. in[17], developed a four-way video conference system. This application is based on a Distributed Mini-MCU running in home servers. The signaling is implemented with SIP and the video is encoded in MPEG-4. Feldmann et. al., in[18] proposes some extensions to the European FP7 project 3DPresence. This project aimed at building a multi-user 3D tele-conferencing system. The system creates life-size video, recognizes gestures and eye contact using depth map and other techniques to create a realistic virtual table. In[29], Davidset. al. present an ongoing web conference application that is based on SIP for session management. Using an adaptation mechanism they create a web-server for SIP signaling over HTTP to establish bidirectional connections using WebSockets.

With respect to the evaluation of WebSockets implement ations we have evaluated on a smartphone, using a variety of technological implementations, the resources consumed by a web application as it displayed a 3D animation stream on the browser[20]. The tested implementations included Ajax and WebSockets and were based on HTML5 features. The experiments showed that the WebSockets implementation consumed the less network and memory resources. In the same context, Gutwin et. al. in[21] compared three different network implementations in the aspect of performance. They compared simple Comet mechanisms, WebSockets and a plug - in approach. Considering the plain browser implementations, Ajax techniques induced a delay of 67ms for LAN connection and 185ms for WAN, but using the

WebSockets implementation the LAN delay reduced to 11ms and the WAN to 86ms.

3. Architecture for Real Time Communications over the Web

In this paper we present a blend of all above features and technologies to introduce a generic architecture for real time communications over the web that can host powerful, collaborative, web applications. In total, our architecture provides web-applications based fully on browsers, with no requirement for any plug-in installation or session management protocol. Each peer client can stream video data from his web-camera and receive video data from a remote camera in real time. The architecture comprises a mediating WebSockets server that listens for incoming connections and the client browsers that should be compatible with HTML5 and WebSockets. At first, an HTML5 peer client requests a web-page from a web server that includes the required JavaScript for media access and streaming over WebSockets. Using the Media Capture and Stream API (GetUserMedia) the camera is captured and image data, in form of JPEG images, are persistently displayed on a canvas. Then, the peer client connects to the WebSockets server using a WebSocket connection and starts the process to send the video data from the canvas to the server. The WebSocketsserver implements session management, hence, it is responsible to bridge the data transferredvia a WebSocketfrom one client peer to others and vice versa. To this end, it maintains an array with all active WebSocket sessions to forward the messages among the involved peers. Each peer client receives and displays the video on another canvas element.

Figure 1 illustrates the components of our proposed architecture along with the involved signaling and data flows. The HTTP signaling between client peers and web server assumes access to the appropriate web page by each client peer. The web page includes the required JavaScript code for drawing a canvas and displaying on it the images flow captured from the web-camera of the peer. In addition it establishes a WebSocket connection with the WebSockets server and streams the captured images from the canvas to it. Finally, the web page includes some buttons for initiating / releasing the data flow to the WebSocketsserver. The platform independent nature of HTML5 enables our architecture to also run on any smartphone device with an HTML5-compliant browser. We have successfully run our tests over Firefox for Android.

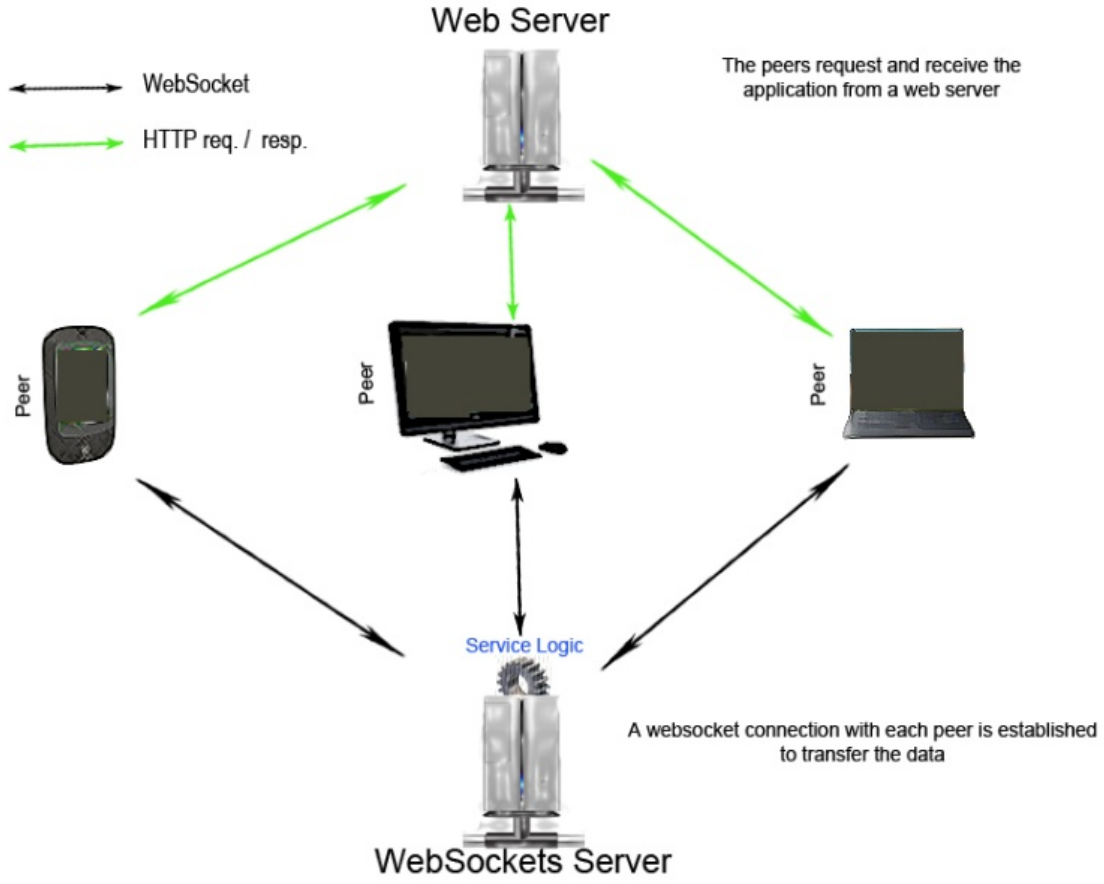

Figure 1. Proposed architecture for realtime video communications over the web

3.1. The WebSockets Server

The WebSockets technology provides a bidirectional communication channel using a single TCP connection. It is designed to be implemented in browsers and web-servers and its API is being standardized by the W3C. The connections are established over the regular TCP port 80, which ensures that the system can run behind firewalls. The life-cycle of a WebSocket session is depicted in Figure 2. At first the client, a browser that supports the WebSockets protocol, requests to establish a WebSocket connection. The positive response from the server denotes the start of such a WebSocket connection. The connection remains open for the whole session, until any endpoint requestsits release with the specified procedure. As a WebSocketremains active; WebSocket frames can be transferred from server to client and vice versa with no preceding request. In our implementation the WebSockets server also hosts the service logic part of our web-applications, which is responsible for maintaining a listing of the client peers with active WebSockets and session management. The service logic is analyzed in the following section. Although logical separated, the web server, the service logic and the WebSockets server could run on the same physical entity. In our implementation we have used Java and Jetty[22] libraries to run the WebSockets server.

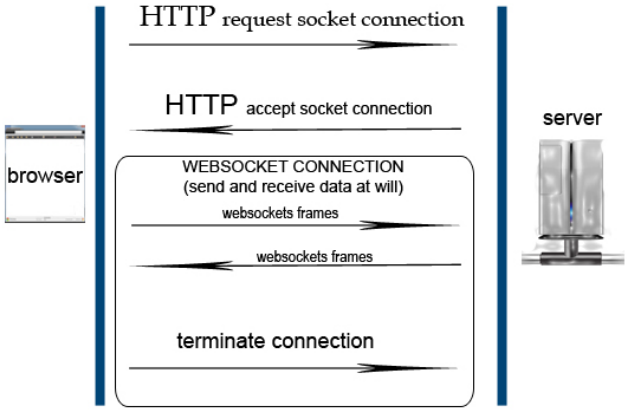

Figure 2. Life-cycle of a WebSocket session

4. Web Applications for Real Time Video Communications

Although the above architecture is generic enough to fit several communication scenarios, we demonstrate it in this paper via two web-applications: one for live video streaming and one for video chatting. Obviously, such applications can find applicability in various domains including tele - education, gaming, entertainment, broadcasting/ multicastin g, virtual meetings. Taking into account that it does not require any plug-in installation or session protocol support, it can definitely be the most convenient solution provided so far to this end.

4.1. Live Video Streaming over the Web

This application involves one streamer (who streams his camera to an audience via the web), one or more receivers (who receive at their browsers the media stream of the streamer) and the service logic (which orchestrates the communication). The application is available via a URL common for both streamers and receivers. The scenario assumes that any available stream can be either a broadcast or a multicast event. Each event is associated with a unique identifier that is called "group name". Broadcast events are available to anyone; hence their group names are propagated by the web server to anyone interested to follow them, while multicast events are close ones restricted to those knowing the associated group names. The interactions that take place are analysed as follows:

Figure 3. Web application for live video streaming

A client peer is connected via its HTML5- and WebSockets- compatible browser to the URL of the application. The web-page requests the user to enter the "group name" of the multicast event he wishes to follow or to select one group name from the list of the available broadcast events. If the user enters a group name that has not been registered previously with the web server, the server assumes it is a new streaming event and registers the user as "streamer". Then, the user is asked to specify if it is a multicast or a broadcast event. In the latter, the streamer should also specify the starting time for the stream. When the streamer submits this info to the server the JavaScript code for the streamers runs at his browser and a WebSocket connection to the WebSockets server is established. Along with this standard signaling, the group name of the event is transferred with a message via the WebSocket to the service logic running on top of the WebSockets server. The service logic registers the specific WebSocket as a streamer WebSocket and associates it with the received group name. Now the streamer can propagate the group name of this available multicast stream to his audience. Whenever a client peer connects to the URL of the application and enters the group name of this multicast, the JavaScript code for the receivers runs at his browser and he is being connected to the WebSockets server via a WebSocket. The WebSocket is registered at the service logic as a receiving WebSocket. In parallel, the group name of the event is transferred to the service logic with a message via the WebSocket. The group name is used by the service logic to match the receiver with the correct streamer WebSocket, so forwards the correct video frames to it. Hence, the receiver displays at a canvas in his browser the stream from the streamer. The same procedure is repeated every time a receiver is connected to the WebSockets server for the specific stream. In case a user selects to follow a broadcast event, the procedures are the same and the receiver is connected to the streamer WebSocket that corresponds to this broadcast group name. It is assumed here that any streaming event registered as a broadcast one is included by the server to the list of such events. Hence, propagation of a broadcast event to an audience by its streamer is optional. Both streamer and receivers have the option to terminate their session (that is their WebSocket) at any time. Figure 3 illustrates the initial web-page for the streamer (a) and for the receivers (b) of our web application, the display of the streamer (c) and the display of the receiver (d).

4.2. Web Video Chatting

This web application is quite similar to the aforementioned one as it concerns the general scenario. A web user who wishes to initiate a video chat session with another web peer visits the URL of the application and enters the group name of the chat. If the given group name is not previously registered with the application it is considered as a new chat session and the user is marked as the "initiator" of the session. Then the appropriate JavaScript run at his browser and an "initiator" WebSocket is established with the WebSockets server. The group name for the chat is also transferred via the WebSocket to the service logic of the application, which associates the specific initiator WebSocket with this. Then, the initiator can inform his chat peer for this available group name. When the second peer connects to the URL of the application and enters the group name of this chat, the appropriate JavaScript connects his browser with the WebSockets server and transfers the given group name to the service logic. The WebSocket of the second user is matched via the group name with the initiator WebSocket and hence, the video frames from the initiator peer are transferred to the second one via the WebSockets server and vice versa. Both initiator and second peer have the option to terminate their session (that is their WebSocket) at will. Figure 4 illustrates the initial web-page of our web application asking the users to determine the group name of the chat (a), (b), the display of the initiator (c) and the display of the second peer (d).

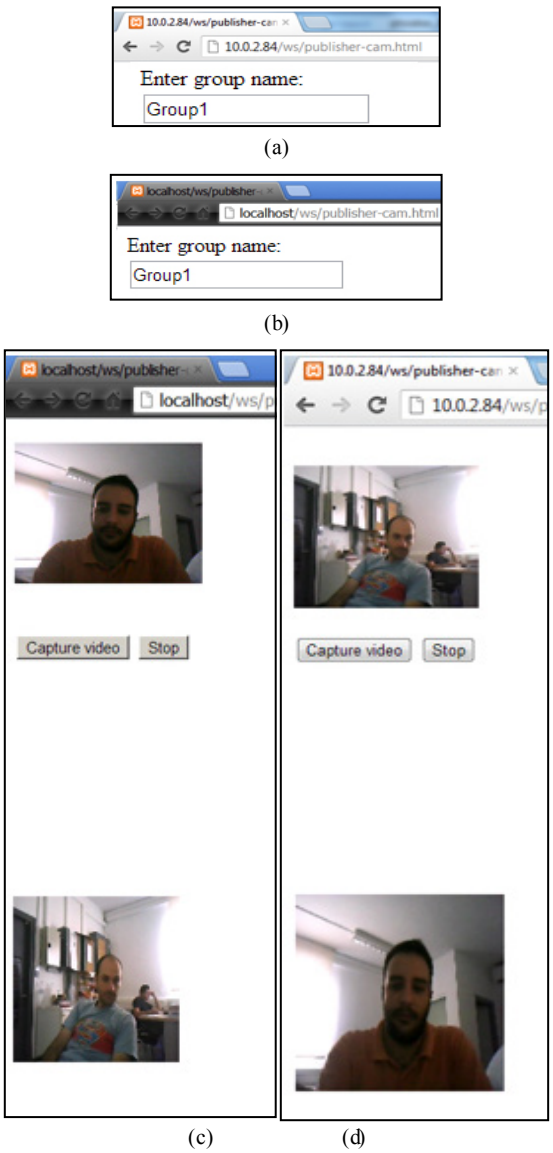

Figure 4. Web application for video chatting

Definitely, such a web application for video chat between

two peers could have been implemented via WebRTC, which additionally assures peer-to-peer transfer of video frames between the peers, that is, without the mediation of a WebSockets server. However, the web application demonstrated here is just the first step towards our goal, which is the implementation of a web-based MCU for the conference-like communication between several peers. Such conferencing requires the mediation of a central MCU justifying thus our implementation approach. In addition, it is worth noticing here that our approach is simpler than WebRTC, since it only requires the support of HTML5 and WebSocket protocol by the clients. Session management here is implemented by our service logic without the requirement for JSEP or other session description protocol by the peers.

4.3. The Service Logic

The service logic for both applications runs on top of the WebSockets server (Figure 1) and in our development is responsible for implementing the communication between the client peers. In particular, it receives the messages sent by one client peer over one active WebSocket, and delivers them to its connected client peers over the corresponding WebSockets. Hence, the service logic is responsible for implementing session management in our architecture. To this end, a java connection-object that holds the data associated with each active WebSocket is created. In addition, an Array List correlates the connection objects for all active sessions. The Group name transferred by each peer within its WebSocketis used as the key for session identification and peers correlation. Hence this Array holds the participants in each session. To realize service logic we are overwriting methods such as the onOpen, onMessage and onClose on the server. In particular, the onOpen method is triggered whenever a new WebSocket connection is established. As opposed to this action, the onClose method is triggered whenever a request for releasing a WebSocket arrives and, hence, the ArrayList object is updated accordingly. The onMessage method is triggered whenever a new message arrives from a client peer. Messages from one client peer are forwarded to all active client peers in the session.

4.4. Our Test-bed

To set up our test-bed we used the Jetty-8 server for WebSockets implementation. Our desktop client peers run Chrome browser (Version 22.0.1229.79 m) and our mobile one (an Android LG P970 Smartphone) a Firefox browser (Version 15.0.1). The service logic has been implemented in Java, on top of the Jetty-8 WebSockets server, using the eclipse IDE. We used HTML web pages with embedded JavaScript code for capturing media streams and opening WebSockets and an apache web server to deliver them on request. The main technologies we used for the applications include: the getUserMedia method for capturing the media streams when required, WebSockets for the communication between client peers via the service logicand Ajax for updating the DOM and reconstructing the web page at client side according to the requests. Our web applications run equally on desktop and portable environments. As Figure 5 depicts, we use a laptop computer as streamer and a variety of computers and mobile devices as receivers. Figure 6 illustrates a web chatting session between two portable devices.

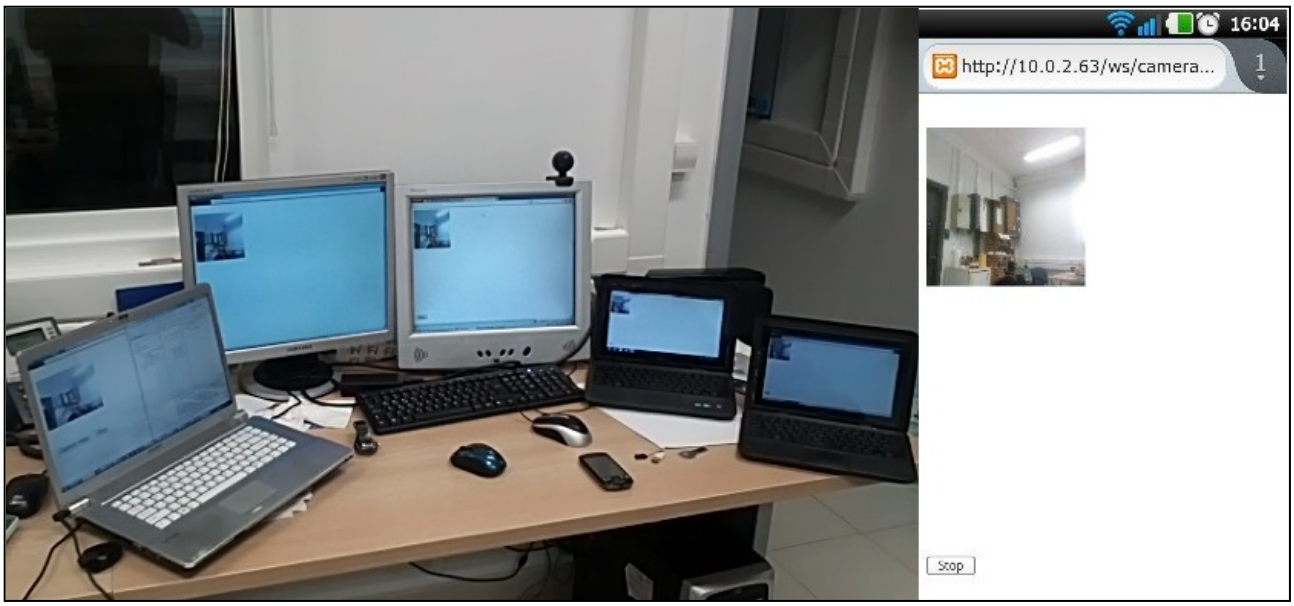

Figure 5. Test-bed for the streaming application

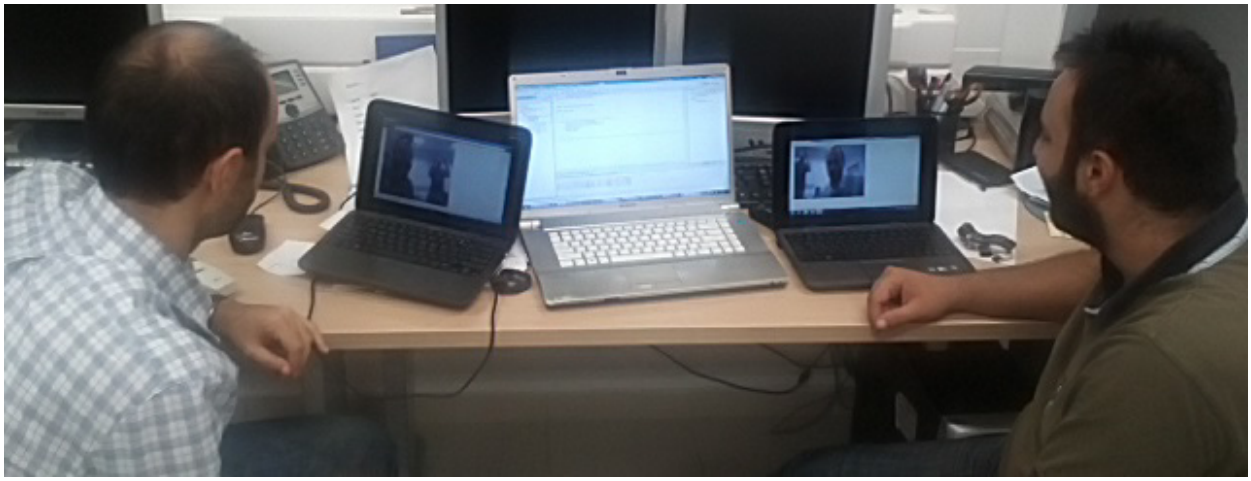

Figure 6. Test-bed for the chatting application

5. Conclusions

A web application using HTML5 and WebSockets technologies can run not only on any operating system, but also on any device with an HTML5- and WebSockets-compliant browser. In this paper we presented a generic architecture that implements HTML5 and other new technologies recently introduced with HTML5. In particular, the canvas element supports video display by a camera and graphics design on the fly. Furthermore, the WebSockets protocol establishes a bidirectional connection between a server and a browser. Taking advantage of these technologies, we demonstrated two cross - device, collaborative web applications. Both applications use the browser to establish a communication with a web-camera. The data from the camera is streamed over a WebSocket connection to a server and is finally delivered to the connected clients. The first application enables live video streaming to all connected users and the second a web chat between two users.

Our approach is simpler than other similar attempts (e.g. the WebRTC), since it only requires the support of HTML5 and WebSocket protocol by the client peers. Session management is implemented by our service logic without the requirement for any session description protocol by the peers.

Our future plans include the stress test of our test-bed to optimize its performance and make it able to handle as many simultaneously connected users and sessions in both applications. So far our system has run with up to 10 client-peers without serious performance degradation, but further testing with more active peers and parallel sessions is required.

Additionally, we plan to evolve the server logic of our web chatting application to an optimal and fast web MCU that will be able to manage conference groups of three or more client peers.

As the above technologies mature and their adoption in commercial browsers increases, the development alternatives for such applications will vary. This can help in more optimal implementations. For example, in our implementations we are repeatedly capturing pictures from the camera of a client peer which are then drawn on a canvas before streaming in a WebSocket. The very fast iteration of this procedure is that offers the illusion of video streaming. Definitely, this is a heavy procedure for both client and transmission. But at the time of our implementation this was the only available solution. When the direct capturing and streaming of media from the camera to the WebSocket is enabled, the intermediate step of drawing on the canvas will be obsolete and the whole procedure will be lightened.

REFERENCES

[1] The Session Initiation Protocol (SIP), IETF RFC 3261.

[2] The Extensible Messaging and Presence Protocol (XMPP), IETF RFC 6120.

[3] HTML5, http://www.w3.org/html/wg/drafts/html/master/ (as visited on 6/22/2013).

[4] WebGL, https://www.khronos.org/registry/webgl/specs/1.0/ (as visited on 6/22/2013).

[5] SVG, http://www.w3.org/TR/SVG/ (as visited on 6/22/2013).

[6] Quartz 2D, http://en.wikipedia.org/wiki/Quartz_2D (as visited on 6/22/2013).

[7] Media Capture and Streams, http:// www. w3. org/ TR/ mediacapture -streams/ (as visited on 6/22/2013).

[8] Device APIs Working Group, http://www.w3.org/2009/dap/ (as visited on 6/22/2013).

[9] The WebSocket Protocol, IETF RFC 6455.

[10] The WebSocket API, http://dev.w3.org/html5/websockets/ (as visited on 6/22/2013).

[11] WebRTC 1.0: Real-time Communication Between Browsers, http://dev.w3.org/2011/webrtc/editor/webrtc.html (as visited on 6/22/2013).

[12] Javascript Session Establishment Protocol (JSEP), draft-ietf-rtcweb-jsep-03.

[13] Ericsson LABS BLOG about WebRTC, visited on 3/16/2013,“ https://labs.ericsson.com/blog/web-real-time-communication-api---the-next-step “

[14] Daoust François, "Adopting HTML5 for Television: Next Steps ", in proceedings of 2011 NEM Summit, Torino, Italy, September 27-29, 2011.

[15] Daoust François, Philipp Hoschka, Charalampos Z. Patrikakis, Rui S. Cruz, Mário S. Nunes, and David Salama Osborne, "Towards Video on the Web with HTML5", NEM Summit, Barcelona, Spain, Oct. 13-15, 2010.

[16] Cazenave, Fabien, Vincent Quint, and Cécile Roisin, "Timesheets. js: When SMIL meets HTML5 and CSS3" , in Proceedings of the 11th ACM symposium on Document engineering, pp. 43-52. ACM, 2011.

[17] Han Intark, Hong-Shik Park, Young-Woo Choi, and Kwang-Ro Park, "Four-way video conference and its session control based on distributed mini-MCU in home server ", in proceedings of IEEE International Conference on Consumer Electronics (ICCE 2008), pp. 1-2, 2008.

[18] Feldmann I., O. Schreer, P. Kauff, R. Schäfer, Z. Fei, H. J. W. Belt, and Ò. DivorraEscoda, " Immersive multi-user 3d video communication", in Proceedings of International Broadcast Conference (IBC 2009), Amsterdam, Netherlands. 2009.

[19] Davids Carol, Alan Johnston, Kundan Singh, Henry Sinnreich, and Wilhelm Wimmreuter, "SIP APIs for voice and video communications on the web", in Proceedings of the 5th International Conference on Principles, Systems and Applications of IP Telecommunications, ACM, 2011.

[20] Kapetanakis Kostas, and Spyros Panagiotakis, "Evaluation of techniques for web 3D graphics animation on portable devices”, in proceedings of the the IEEE International Conference on Telecommunications and Multimedia (TEMU 2012), pp. 152-157. July-August 2012.

[21] Gutwin Carl A., Michael Lippold, and T. C. Graham, "Real-time groupware in the browser: testing the performance of web-based networking", in Proceedings of the ACM conference on Computer supported cooperative work, pp. 167-176, 2011.

[22] The Jetty project, http://www.eclipse.org/jetty/about.php (as visited on 6/22/2013).