2012 Third FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing

Real-Time Collaborative Video Watching on Mobile Devices with REST Services

Juan (Susan) Pan
Computer Science Dept, NJIT
Newark, NJ 07102, USA
Email: jp238@njit.edu

Li Li
Avaya Labs Research
Basking Ridge, NJ 07920, USA
Email: lli5@avaya.com

Wu Chou
Avaya Labs Research
Basking Ridge, NJ 07920, USA
Email: wuchou@avaya.com

Abstract— **In this paper, we describe a mobile application that synchronizes the playback of a video stream among a group of mobile devices while the video playback is controlled (e.g. pause, resume, seek) by one of the devices. This feature allows a group of people to collaborate remotely in real-time through watching the same video on their mobile devices. To make this application work for heterogeneous mobile devices and networks, we develop synchronization protocols and time-based video position prediction algorithms on top of a REST platform that is independent of programming languages, operating systems and transport protocols. A prototype system has been developed based on Android phones connected by 3G cellular network. The preliminary experiments show the approach is feasible and promising.**

Video collaboration; playback synchronization, mobile device; REST services

I. Introduction

Since the invention of digital camera, digital videos (live stream or recorded) have become an important source of information and knowledge that are widely accessible over the Internet. Combined with various forms of social media, video has become a medium for communication, interaction and collaboration between people. At the same time, more and more mobile devices, including smart phones and tablets, with more computing power, advanced sensory functions, and faster wireless network connections, offer API for developers to create brand new applications.

Motivated by these social and technological trends, our goal is to develop a real-time collaborative video watching application that allows people in different locations to collaborate in real-time around the same videos played on their mobile devices. This application could be used in many situations and domains. For example, it could be used by a virtual project team to listen and discuss a technical presentation. It could also be used by a teacher in an interactive virtual classroom to present a subject to the students through a collection of videos. In a multimedia call center, a residence agent could use this application to show customers how to assemble a faucet through some recorded video instructions. Most current video players support start, stop, pause, resume and seek. More advanced control actions include zoom, pan, and rotate that are useful for watching 3D videos. The goal of this application is to synchronize such video playback on multiple mobile devices while the playback is controlled by one of the devices. With perfect playback synchronization, the participants in different locations will be able to watch the same video on their own devices as if they were watching the video on one device together. For example, a moderator can pause the video to explain a scene and then resume the playback. When this happens, the video displays on all other participating mobile devices should be paused and resumed at the same time as the moderator such that the same video context is always shared among them.

The application does not limit how a device accesses the video source: a device can download the video before playback or stream it from the network during playback. The application does not require that all devices start the video playback at the same time either. Instead, the devices can join and leave a playback session at random.

The application is also independent of the device types, device locations and network infrastructure. Devices may have different operating systems and APIs in different programming languages and support different video codecs and playback controls. Devices may be located in the same room with fast and reliable networks (e.g. LAN) or distributed in different places with heterogeneous networks (e.g. 3G and 4G cellular networks).

To achieve perfect synchronization under these situations is a challenge. Even though the devices can play the same video in a particular session, they may play the video at different speed because they have different playback engines and network bandwidth. These differences can be corrected by synchronization messages sent from the control device. But too many messages may drain the power of the devices and overwhelm the network. Furthermore, the synchronization messages may take different time to arrive and these message delays may impact the video synchronization.

To address these issues, we develop a lightweight REST service platform Cofocus that exposes the media control functions in mobile devices as REST services in a logical peer-to-peer network. On top of this platform, we propose video synchronization protocols and algorithms to synchronize video controls among mobile devices with REST services. In addition, we propose protocols for ad-hoc session management and text chat with end-to-end security to facilitate secure collaboration over heterogeneous networks.

The rest of the paper is organized as follows. Section II will review the related work. Section III provides a brief introduction to the Cofocus platform. Section IV is dedicated to the protocols and algorithms of the real-time collaborative video watching application. Section V discusses the security aspects. Section VI presents some preliminary experimental results and we conclude with Section VII.

II. Related Work

Multiparty video conferences are now available on many high-end mobile devices with video cameras. Our application is different from video conference in several ways. First, in video conference, each device receives multiple video streams from the cameras of the other devices, whereas in our application, all devices receive a single video source, which can be from the Internet or a local file on the devices. Second, in video conference, a device can only control its own video views, not those of the other devices, whereas in our application, a control device can synchronize the video views on all other devices. For these reasons, our application is complementary to video conference and can be integrated into video conference as a special mode.

Interactive TV (ITV) [1] is defined as applications and programming that allow the viewer to control content delivered with and through the television. Although ITV supports playback control on delivered videos, one can only control his own television. Although a viewer's action may influence the content delivered to other views, for example by voting, such influences are the result of aggregated actions from many viewers instead from one particular viewer.

Social Television is a general term for technology that supports communication and social interaction in either the context of watching television, or related to TV content [2][3][4]. It incorporates some of the ideas from ITV such as viewer participation. Our work is motivated by this trend and attempts to extend the idea to mobile devices and online videos. However, most work in Social TV focus on how to combine social relations and interactions with TV experience. As far as we know, none of them proposes to allow the viewer to synchronize TV content across televisions.

There are many social media web sites that support social interactions with and around videos, such as livecasting (e.g. www.livestream.com) and video sharing (e.g.YouTube), as well as video capturing and sharing on mobile devices [5][6][7]. However, none of these applications provide the means for a user to control the playback of a video for a group.

Shen et al [8] proposes a system and algorithms to split a high-resolution video stream among 2 mobile phones arranged side-by-side to form a composed screen. Although this system uses protocols and algorithms to synchronize the half frames between the phones, it does not synchronize user playback control across the phones. Moreover, the approach assumes the phones are homogeneous and placed in close proximity, which are not required by our application.

There are some prototype systems to support real-time collaborative video annotations [9][10][11][12] on desktop computers. In these systems, a group of users can annotate and share their annotations about the videos that they are watching in real-time. Although synchronized video playback seems a useful feature to these systems, these papers do not present means to synchronize playback controls across devices.

Google Plus also allows real-time synchronization of shared YouTube videos with its Hangouts tool on mobile phones [17]. Anybody in a Hangout can start, stop, pause, and resume a video and the action will be reflected in all the web browsers in the Hangout. However, Google Plus only plays YouTube videos whereas our application works for videos from other sources. The Hangout tool requires a Google Plus account to watch a video, whereas our application works for any XMPP account. Furthermore, the Hangout tool depends on a central Google Plus service whereas our application is peer-to-peer based. In our application, the application states, such as video link, sessions, and participants, are stored in the peers instead of a central server.

III. Cofocus Rest Platform

We choose REST as the architectural style of our mobile service platform for several reasons. First, REST encourages independent development of components in a distributed hypermedia system with flexibility, usability, simplicity, efficiency, scalability and extensibility. Second, REST services are easy to integrate with the Web. In our approach, each mobile device acts as a web server that exposes its functions as REST services. At the same time, it acts as a web client that accesses the services using the HTTP protocol.

However, there are some issues to adopt REST for real-time collaborative applications in a mobile environment as summarized below:

1. A mobile device does not have a reachable IP address to act as a web server.
2. REST lacks mechanism to support asynchronous message exchange patterns that are important for event-driven applications.
3. The full-fledged HTTP is too heavy for resource constrained mobile devices.
4. REST does not have built-in support for presence which is important in collaborative applications.

To address these issues, we proposed a Compact HTTP over XMPP [13] protocol to support peer-to-peer asynchronous interactions between mobile devices. In this architecture [14], a REST server becomes a XMPP entity identified by a JID. The resources of a REST server are identified by the URI template http://xmpp:{jid}/{path}. Since a REST server is a XMPP entity, a mobile device can know in real-time when a REST server joins or leaves a collaboration session based on the presence information provided by the XMPP layer about XMPP entities. With this approach, a group of mobile devices can form an ad-hoc collaborative mobile web in which devices can join and leave at random.

Figure 1 illustrates a small ad-hoc web formed by 3 mobile devices (phone images) that joins the rest of the Web represented by two computers. The dashed arrows indicate the messages sent from REST clients to REST servers. In this system, the web browser can access the resources on the ad-hoc web. Conversely, the ad-hoc web can access the resources in any web server. Collaborative video watching is one of the applications developed based on this platform.

IV. Collaborative Video Watching

To give the users control over their mobile devices, the life cycle of this application consists of the following phases that can be controlled by a user through the GUI:

1. **Application startup**: the moderator and the participants start up the REST services on their mobile devices and login the XMPP network using their JID (Figure 2).
2. **Session setup**: the moderator invites the participants to join the session. The invite message specifies a video playlist and the start date/time. Upon receiving the invite message, a participant can join the session right away or wait until a later time (Figure 3).

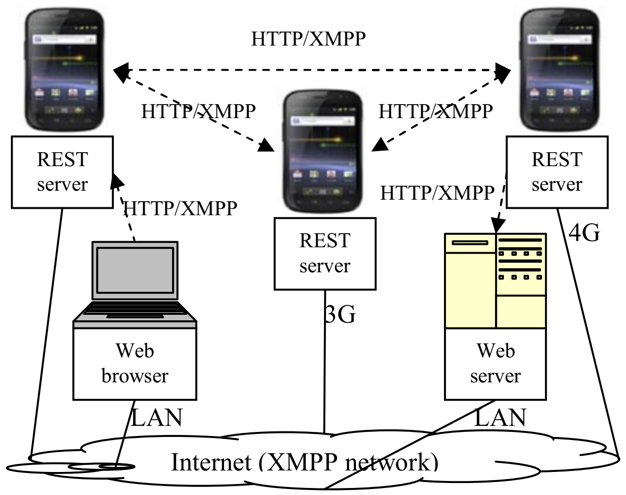


Figure 1: Ad-hoc web formed by mobile devices

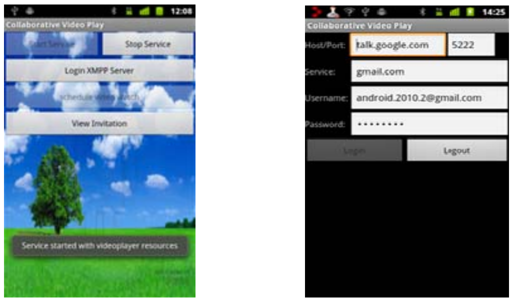
3. **Collaboration**: when the schedule time is up, the video playback on the joined mobile devices will start automatically at the same time. During this phase, the moderator can control the video playback at will, such as pause/resume/seek/reset, and the video views on the participating mobile devices are synchronized automatically with moderator's device. A moderator can also select different video to play and monitor the video progress on other mobile devices. A participant can monitor moderator's video progress. A participant can join and leave an active session at any time without affecting other participants' video views. Whenever a participant joins, his video view will be synchronized automatically with the current view of the moderator. Participants and the moderator can also exchange text chat messages (Figure 4).
4. **Session termination**: the moderator sends the termination messages to all the invited participants and terminates the session. No participant can join the session anymore.
5. **Application shutdown**: the moderator and the participants log out the XMPP network and stop their REST services. No session can be created anymore.

Phases 1 and 5 give the users control over the REST services and the application on their phones for security reasons and power conservation. Phases 2 and 4 allow a moderator to create many sequential sessions within one application. This also increases security as the moderator can secure each session differently, for example with a new security token. The current implementation only supports text chat because the smart phones that we experiment with do not support voice call and video play at the same time. The video progress bars allow a user to check how far apart the video playbacks are and make necessary adjustment if possible.

The video playback synchronization in phase 2 is the core of this application. Ideally, we should synchronize the video frames across the mobile devices to achieve precise alignment. However, we do not take this approach for the following reasons. First, this approach requires low-level access to the video decoders and playback engines on mobile devices. As far as we know, none of the mobile device APIs on the market offers this access. Even though we can modify the required components if they are open source, a user has to upgrade their system to use our application. Second, some mobile API, such as Android SDK [15], offers time-based video playback controls that are good enough for our application.

Synchronizing the event across mobile phones often requires the clocks on the devices are synchronized or has bounded drift. However, this is out of the scope of our work since there are various solutions, such as NTP [16], to address this issue. In this paper, we assume the clocks on the mobile devices are synchronized.

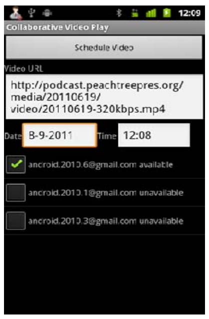
Our application only considers a time-based video playback model proposed by Android SDK which is very common for web based video playback control. In the time-based model, a video (file or stream) is divided into a series of time-based positions that is 1 millisecond apart. Each playback control action moves the video from or to one of these positions.



Host/Port: talk.google.com 5222
Service: gmail.com
Username: android.2010.2@gmail.com
Password: ••••••••

Figure 2: Start Service (left) and login to XMPP (right) for both moderator and participants

To relate the time positions to video frames in this model, we assume the video frame rate on the mobile devices is the same *R* fps. This is a reasonable assumption because all devices play the same video. If the video on mobile device 1 is at position p_1 and the video on mobile device 2 is at position p_2, then the minimal frame difference between the two mobile devices is $\left\lfloor \frac{R|p_1-p_2|}{1000} \right\rfloor$. Our goal is to reduce the position difference with minimal number of messages.



Video URL

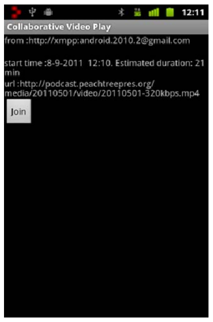
http://podcast.peachtreepres.org/media/20110619/video/20110619-320kbps.mp4

Date 8-9-2011 Time 12:08

[✓] android.2010.6@gmail.com available

[ ] android.2010.1@gmail.com unavailable

[ ] android.2010.3@gmail.com unavailable

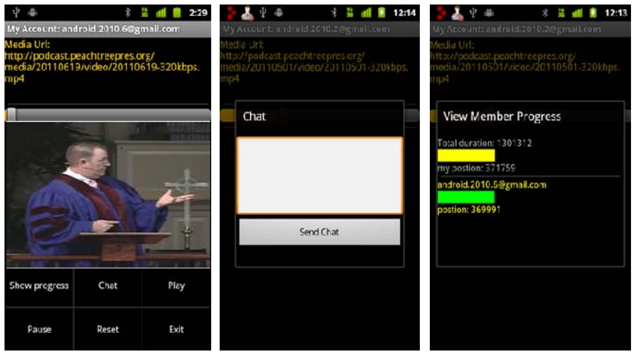


from :http://xmpp:android.2010.2@gmail.com

start time :8-9-2011 12:10. Estimated duration: 21 min

url :http://podcast.peachtreepres.org/media/20110501/video/20110501-320kbps.mp4

Figure 3: Invite participant by moderator (left) and accept invitation by participant (right)



My Account: android.2010.6@gmail.com

Media Url: http://podcast.peachtreepres.org/media/20110619/video/20110619-320kbps.mp4

My Account: android.2010.2@gmail.com

Media Url: http://podcast.peachtreepres.org/media/20110501/video/20110501-320kbps.mp4

My Account: android.2010.2@gmail.com

Media Url: http://podcast.peachtreepres.org/media/20110501/video/20110501-320kbps.mp4

Total duration: 1301312

my postion: 371759

android.2010.6@gmail.com

postion: 369991

Figure 4: Video control by moderator (left), chat window (middle) and video progress (right)

The simplest synchronization with minimal number of message is for the participant devices to duplicate the moderator's control actions. But this approach ignores the position difference caused by network delay. For example, when a moderator pauses at p_1 and the network delay is $d>0$, then the duplicated pause at a participant will be at p_1+d. For $R=24$ fps and $d=600$ milliseconds, the participant video view will be 14 frames late which may be a completely different image from the moderator's frame. To compensate these delays, a participant device needs to predict the moderator's position based on its current position, playback speed, network delay and execution delay. The network delay occurs when synchronization messages are transmitted between devices and the videos are downloaded to devices over the network. The execution delay occurs when playback control threads take time to complete on a device.

To support position prediction, a moderator's control action is conveyed to a participant by a 4-tuple (a, p, t, s), where a is a control action, p is the video position related to the action, t is the timestamp (in millisecond) when the position is obtained, and s is the playback speed which is the number of video positions advanced by a device in one second clock time. $s<1$ when the video playback is slower than real-time; $s=1$ otherwise. It can be estimated by the moderator device for each control action.

The playback control actions are implemented based on Android SDK. They are executed by the devices in reaction to user input or synchronization messages:

- *prepare()*: prepares the video for playback;
- *start()*: starts the video playback at the current position and returns that position;
- *pause()*: pauses the video playback at the current position and returns that position;
- *seek(p)*: changes the current video position to p;
- *get()*: returns the current video position;

The following subsections describe the synchronization protocols and position prediction between a moderator device and a participant device. The protocols can be generalized to multiple participant devices using XMPP group chat service such that the number of messages sent from the moderator device will remain the same when the number of participant devices increases. For clarity, the synchronization HTTP messages between devices are encoded as n-tuple.

A. Scheduled Start

The invite protocol and the start protocol are illustrated in Figure 5. The invite protocol contains two messages, invite and join, that set the scheduled time on all the devices.

When the scheduled time occurs, all devices prepare the video playback buffer automatically. The *(prepare, complete)* message informs the moderator that the participant device has finished the preparation of video playback. When all the participant devices are ready, the moderator device starts the playback for itself and the participant devices. This two step approach eliminates the preparation variations in the mobile devices.

If the moderator device D_1 is at position p1 on timestamp t_1, then the delay (in millisecond) for D_2 to process this control includes: 1) execution delay d_1 at D_1 to send the control message; 2) network delay d_{12} between D_1 and D_2; 3) execution delay d_2 at D_2 to receive the control message. To compensate these delays, D_2 should start at position $p_2 = p_1 + s(d_1 + d_2 + d_{12})/1000$. This means D_2 has to seek forward to p_2 and the delay d_s cause by this operation should also be compensated. On the other hand, delays in 1) and 3) are insignificant and can be ignored. If D_2 receives the control message at timestamp t_2, then $d_{12} = t_2 - t_1$. With these adjustments, the predicted video position on D_2 is $p_2 = p_1 + s(d_s + t_2 - t_1)/1000$, where d_s is estimated by D_2.

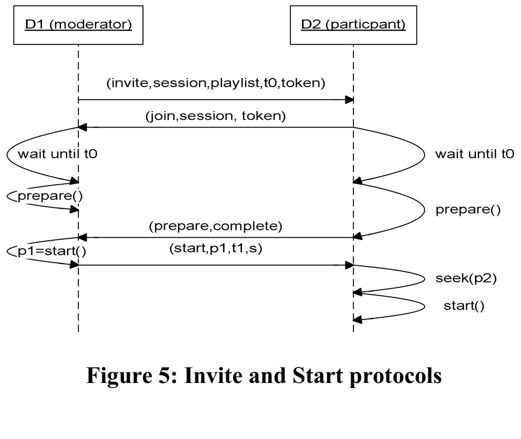


Figure 5: Invite and Start protocols

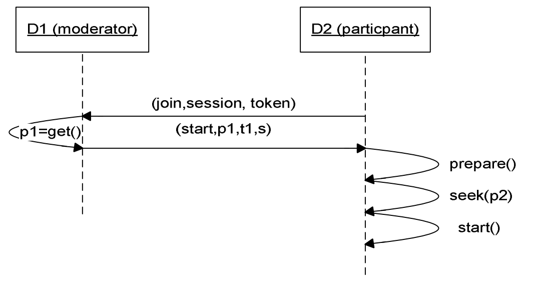


Figure 6: Join protocol

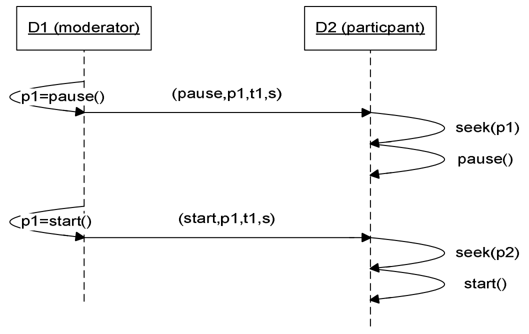


Figure 7: Pause and Resume protocols

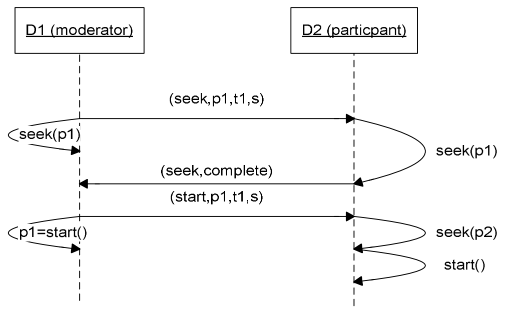


Figure 8: Seek protocol

B. Midway Join

A participant can join a session while the video playback is in progress. The participant device sends a join to the moderator device and the moderator device responds with a start tuple (Figure 6). D_2 calculates its position in a way similar to section 4.1 except that D_2 has to compensate for the video prepare time d_p, which is estimated by D_2. So the estimated video start position on D_2 is $p_2 = p_1 + s(d_p + d_s + t_2 - t_1)/1000$.

C. Pause/Resume

The moderator can pause and resume a video at any time (Figure 7). Whenever D_1 pauses its video playback, D_2 should pause its video playback and seek backward to the same position. When D_2 resumes, a start tuple is sent from D_1 to D_2 which starts at a position as calculated in section 4.2.

D. Seek

The moderator can seek forward and backward to any position and the video playback will start automatically forward from that position (Figure 8). Seek control may take some time to complete depending on the target position and the video streaming speed. For this reason, D_1 first instructs D_2 to seek. After D_2 completes, D_1 instructs it to start playback at the new position. D_2 calculates the new position p_2 as described in section 4.1 to compensate the delays as the seek operation is similar to the prepare operation.

V. SECURITY

Because a mobile device may contain personal and confidential information, any collaborative application must have security mechanism to protect this information. Our application relies upon several layers of security mechanisms: network, service and application.

In the network layer, we use XMPP to authenticate XMPP entities and authorize message exchanges. XMPP accounts are protected by passwords to prevent unauthorized access and JID spoofing. A XMPP entity can elect to receive messages only from its trusted buddy list to reduce the risk of attacks. XMPP messages can also be encrypted with TLS to ensure confidentiality.

In the service layer, the Cofocus platform allows a user to specify user-based access polices on his device to prevent unauthorized access. The platform also supports symmetric key encryption for end-to-end message encryption in case some XMPP servers do not support TLS or the message traverse different networks.

In the application layer, security token are used for session startup and join to ensure that only devices with the authenticated token can participate in the session.

VI. IMPLEMENTATION AND EXPERIMENTS

A prototype system with the proposed architecture and synchronization protocols is implemented as an Android application and tested on two Google Nexus S 4G phones running Android SDK 2.3. The phones subscribe to the Sprint 3G cellular networks and connect to the public Google Talk XMPP server (talk.google.com). The main components of the application are illustrated in Figure 9. The application consists of GUI classes that interact with the REST resources managed by the Cofocus platform. The resources control the media player and manage sessions based on the HTTP messages sent over the XMPP transport. We used public videos posted on the Web for testing purpose.

The first experiment evaluates the worst-case delay when the video playback is moved forward and backward at random. In this experiment, we measure the synchronization completion time $d_c = t_e - t_s$, where t_s is the time when a control action is received at the moderator device and t_e is the time when the corresponding seek is complete on the participant device. The test MPEG 4 video is 26 minutes long with 64 MB size (http://podcast.peachtreepres.org/media/20110501/video/20110501-320kbps.mp4). 154 seek operations alternating between the near start and near end of the streamed video are performed and the statistics of the completion time are summarized in Table 1.

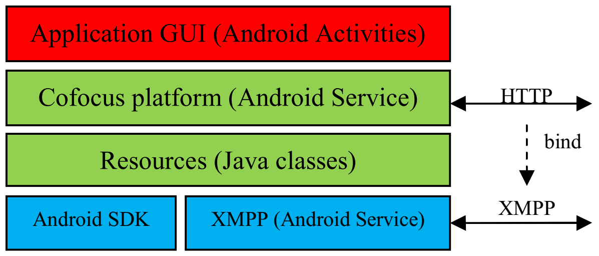

Application GUI (Android Activities)
Cofocus platform (Android Service)
Resources (Java classes)
Android SDK
XMPP (Android Service)
HTTP
bind
XMPP

Figure 9: Main components of application

Table 1: Synchronization Completion Time

Statistics	Synchronization completion time (millisecond)
Q1	10239
MIN	8445
MEAN	11547
MAX	15815
Q3	12728
STDEV	1666

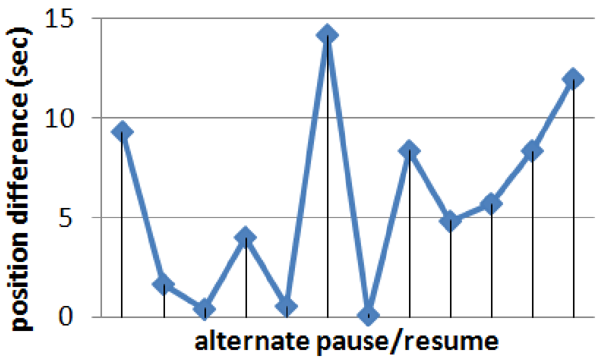


Figure 10: Position differences under playback controls

The experiment shows that on average, it takes about 11 seconds to complete the synchronization when the video playback moves across positions that are 26 minutes apart. The synchronization time is about 0.7% of video duration for the seek control based on this experiment.

The second experiment measures the video synchronization under playback controls. In this experiment, during the video playback, the moderator alternates pause and resume actions 6 times where the pause actions are about 1 minute apart and the resume is about 20 seconds after the pause. After each action, the current video positions on the two phones are read from the video progress screen (Figure 4 right). The chart in Figure 10 plots the position difference for each action, where the average difference is 5.7 seconds with a 4.7 seconds standard deviation.

VII. CONCLUSIONS

We briefly introduced the REST platform on which this application was developed and presented the novel protocols and video position prediction algorithms for video synchronization. The prediction algorithms can estimate the target position to tolerate network and control delays. This makes the application work on mobile devices with different playback and network speed. A prototype system based on Android SDK was implemented and tested on two Nexus S 4G phones. The preliminary experiments showed the proposed approach is feasible.

REFERENCES

[1] http://www.itvalliance.org/.

[2] Geerts, D. and D.D Grooff. 2009. Supporting the social uses of television: sociability heuristics for social tv. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems* (CHI'09), pp 595-604.

[3] Mitchell, K. et al. 2010. Social TV: Toward Content Navigation Using Social Awareness. In *Proceedings of EuroITV2010,* pp 283-291.

[4] Mantzari, E. et al. 2008. Social TV: Introducing Virtual Socialization in the TV Experience. In *Proceedings of uxTV'08,* pp 81-84.

[5] Kaheel, A. et al. 2009. Mobicast: A System for Collaborative Event Casting Using Mobile Phones, *In Proceedings of MUM09*.

[6] Multisilta, J. and Suominen, M. MoViE: Mobile Video Experience. In *Proceedings of MindTrek 2009*, pp 157-161.

[7] Bao, X. and Choudhury, R.R. MoVi: Mobile Phone based Video Highlights via Collaborative Sensing, *In Proceedings of MobiSys'10*.

[8] Shen, G. et al. 2007. MobiUS: Enable Together-Viewing Video Experience across Two Mobile Devices. In Proceedings of MobiSys'07, pp 30-42.

[9] Miller, G. and Fels, S. 2010. Collaborative Annotation of Multi-Vew Video. GRAND Forum.

[10] Zhai, G. et al. 2005. eSports: Collaborative and Synchronous Video Annotation System in Grid Computing Environment. In *Proceedings of the 7th International Symposium on Multimedia*.

[11] Motti, V.G. et al. 2009. Collaborative Synchronous Video Annotation via the Watch-and-Comment Paradigm. In *Proceedings of EuroITV'09*, pp 67-76.

[12] Attarwala, A. et al. 2010. Real time collaborative video annotation using Google App Engine and XMPP protocol, In Proceedings of IEEE 4th International Conference on Cloud Computing, pp 738-739.

[13] The XMPP Standard Foundation: http://xmpp.org.

[14] Li, L. and Chou, W. 2011. Cofocus: Compact and Expanded Restful Web Services for Mobile Environments. In *Proceedings of WEBIST 2011*, pp 649-656.

[15] *Android Developers*: http://developer.android.com/index.html.

[16] NTP: The Network Time Protocol. http://www.ntp.org.

[17] http://www.google.com/mobile/+/