Developing WebRTC-based Team Apps with a Cross-Platform Mobile Framework

Kundan Singh
Avaya Labs Research
Santa Clara, CA, USA
singh173@avaya.com

John Buford
Avaya Labs Research
Basking Ridge, NJ, USA
buford@avaya.com

***Abstract*—We present lessons learned in developing cross platform multi-party team applications. Our apps include a range of communication and collaboration scenarios: document and content sharing in a team space, an agent-based meeting helper, phone number dialer via a voice-over-IP (VoIP) gateway, and multi-party call in peer-to-peer or client-server mode. We use web real-time communication (WebRTC) to enable the audio and video media paths in the apps. We use frameworks such as Chrome Apps and Apache Cordova to create apps that can be accessed from a browser, or installed on a desktop, mobile device, or wearable. The challenges and techniques described in our paper related to audio, video, network, power conservation and security are important to other developers building cross-platform apps involving WebRTC, VoIP and cloud services.**

Keywords—HTML5, Apache Cordova, Chrome Apps, WebRTC, Mobile, Cloud, Wearable

I. Introduction

Team apps have features that require multi-way media streaming, multipoint notifications, resource sharing, use of cloud services, security, and operation on a range of devices. Underlying technologies used to implement these features such as WebRTC and client-side certificates have inconsistent support, particularly across mobile platforms. The capabilities are also important for mobile app developers in categories including gaming, social networking, healthcare and education.

Due to the plethora of devices, app developers prefer cross-platform development with a *write-once run-anywhere* model. Prior studies have shown that cross-platform development tools for mobile offer the economies of write-once run anywhere but are challenging in terms of creating a native user experience (UX) or using device specific capabilities and advanced browser-specific features. However, prior work does not cover the technologies important to team collaboration apps.

We have developed a set of cross-platform applications using the Apache Cordova (PhoneGap) [1] hybrid app development framework. These applications are implemented in HTML5 and typically use cloud services from our self-service cloud portal [2][3], including WebRTC (web real-time communication) [4] and WebSocket [5][6].

In this paper we discuss challenges and lessons for cross-platform development relevant to classes of apps that involve real-time communication and cloud-based services. The apps that we describe have important requirements beyond those described in other studies for: real-time multi-party streaming of video using WebRTC, real-time notifications, e.g., using WebSocket, secure access to cloud services, and apps interoperability with browser-based and desktop versions. We address challenges including and beyond these requirements.

These apps include: (1) a team collaboration app for persistent sharing of content with escalation to real-time voice, video or app sharing, (2) a web-based video call/conference service that runs all the app logic in the endpoint, (3) a meeting helper agent which joins the conference bridge during a meeting, automatically creates the meeting summary at the end, and sends it to the participants, (4) three softphones using Voice-over IP (VoIP) services for small or large businesses, and (5) a serverless video phone using local network multicast.

Our work answers these questions important to those who are developing with or evaluating tools for cross-platform mobile app development: (1) how to best realize write-once-run-anywhere for apps that leverage features common to team apps, and (2) what are important takeaways from our range of apps' development experience to help existing and new app developers. We list several challenges and techniques related to audio, video, network, power conservation and security.

Remainder of this paper is organized as follows. Section II contains related work on hybrid apps and WebRTC. Section III describes the apps under consideration. Sections IV and V describe the cross platform development process including for wearable. Sections VI to VIII have our lessons learned on HTML5, WebRTC and mobile apps. Section IX describes power conservation and security of notifications. Section X has our conclusions and future work.

II. Related Work and Background

The leading mobile application stores, Google Play and iOS App Store, each have more than 1M mobile apps [7]. Powerful cross-platform mobile app development frameworks [8] have emerged so that developers can produce hybrid native applications from HTML5 – PhoneGap (Apache Cordova) and Sencha Touch – or cross-platform tools and SDKs, such as Adobe AIR, MoSync, Appcelerator, and Corona.

Malavolta et al. [9] study user perception of cross-platform mobile apps by mining 3M reviews of 11,917 free apps from the Google Play Store. Of the 11K apps, about 500 were developed with cross-platform mobile frameworks, and more than 50% of these with Apache Cordova [1]. Their results

978-1-4673-9292-1/16/$31.00 ©2016 IEEE

show that cross-platform development is most frequently used for data intensive apps and least used for apps with platform interaction, e.g., games and audio. They conclude that hybrid development frameworks are perceived as better suited for data-intensive mobile apps than for dealing with low-level, platform-specific features, and in some categories, native apps are viewed better due to performance and less bugs.

Heitkötter et al. [10] found that Appcelerator based mobile apps appear more native, whereas PhoneGap apps are like web sites. Angulo and Ferre [11] evaluated the user experience comparing natively developed app with an equivalent app built using a cross-platform tool. A native iOS UX is more difficult to obtain using cross-platform tools compared to Android apps.

Several studies have shown the feasibility of cross-platform mobile app development, e.g., RSS reader using Appcelerator [12], geolocation and map APIs using PhoneGap [13][14], or WebSocket with PhoneGap [15]. Cross-platform apps using device features such as accelerometers and multi-touch gesture could be comparable to native versions of the same apps [16].

Unlike previous studies, we focus on real-time communication and collaboration, and the strength of our work is in the recommendations about developer practices, the experience with WebRTC, and the number of apps.

WebRTC [4] refers to the ongoing efforts by W3C, IETF and browser vendors to enable web pages to exchange real-time media streams. A page can capture from local mic and/or camera using getUserMedia as a local media stream abstraction, create RTCPeerConnection, a peer-to-peer abstraction between browser instances, and send a media stream from one browser to another (Fig.1). RTCPeerConnection emits certain signaling data such as session description and transport addresses, which must be sent and applied to the RTCPeerConnection at the other browser to establish a media path. Web pages typically use WebSocket [5] or Ajax over HTTPS between the browser and the web server to do such session negotiations. Browsers can use the web page supplied STUN or TURN servers [4] to discover server reflexive and/or relayed addresses for the media paths that cross network boundaries.

III. Multi-party Team Apps

Team apps involve both active or synchronous modes such as video or teleconference, as well as passive or asynchronous modes such as SharePoint or a wiki. A collaboration app often supports both, e.g., a video call app that can send video mail, or a document sharing app that allows chat with other viewers.

A. Sample Team Collaboration Apps

We have built several team apps listed below and shown in Fig.2 with both active and passive modes. We use WebRTC for the peer-to-peer as well as client-server media paths, and typically use Ajax and/or WebSocket on the signaling paths.

1) Engagement Dialer: It allows dialing out a phone number using the enterprise or cloud VoIP service of Avaya's Engagement Development Platform (EDP) and Aura software suite (Fig.2a). A customer subscribes to the service, distributes this dialer app to its users, and provisions user credentials in the auth-service. The auth-service authenticates the app user, and returns a token that is used by EDP to initiate a phone call

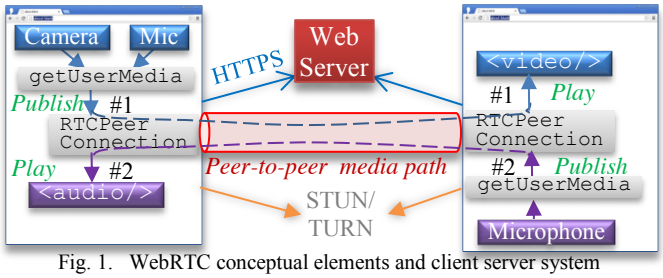


Fig. 1. WebRTC conceptual elements and client server system

using the SIP proxy and gateway. Separating the app user's authentication from the EDP token simplifies billing to that customer, instead of its individual app users.

2) IP office phone: Avaya IP office is a communication system for small and midsize businesses, which we host on cloud for customer trial. The IP office phone is a client to connect to this VoIP system to make or receive voice or video calls. It uses client-server media path anchored at the server (Fig.2b). This is not uncommon in cloud telephony, e.g., for media services of recording, interactive voice response or telephony gateway.

3) Vclick: This is a pure web-based multi-party client to enable video call, conferencing, video presence, text chat, shared white-board, notepad and screen sharing [17]. It does not depend on legacy VoIP systems or browser plugins. It works as a collection of loosely coupled apps that mash up at the data level instead of pair-wise app permissions, e.g., white-board app can be joined from a separate device in an existing desktop video call (Fig.2c). The cross-platform version is created as a single app, with reduced functions of voice, video and text chat, but no other sharing apps. Unlike traditional server-side web apps, Vclick runs all the app logic in the client, and uses a "thin" resource server [6] that enables data storage and pub/sub events, e.g., a client subscribed to a resource gets notified when it changes. A persistent client-server connection enables shared data access and incoming events such as call invite. It creates peer-to-peer (full-mesh) media paths during a call.

4) Connected Spaces: This is a team collaboration system for persistent sharing of content, e.g., documents, meeting notes, wiki or calendar [18]. Long lived distributed groups can organize their work in spaces, e.g., a space for "Third Floor Construction" project contains all the relevant content of that project. Users can create persistent annotations on shared content (Fig.2d) or start impromptu audio, video and/or text chat among the viewers of a document in a space. We have separate web apps for desktop and mobile. The mobile version adjusts to portrait or landscape orientation. It uses HTTPS for all passive collaboration, and secure WebSocket for signaling of active collaboration and real-time notes. It uses Vclick's resource server for such events and data storage, and its WebRTC conversation app for chat. In our cloud deployment, we share the STUN and TURN servers [4] among various WebRTC projects: Vclick, IP office and Connected Spaces.

5) Suki App: Suki meeting helper agent joins the conference bridge during a meeting, automatically creates the meeting summary at the end, and sends it to the participants. The meeting summaries are then available from the Suki app. Although the Suki agent joins the conference in active mode, the client app is passive – for viewing past meeting summaries. The app gets the user's calendar to show the meetings (Fig.2e).

6) *SIP in JavaScript*: Unlike Engagement Dialer or IP office phone that run the signaling protocol in the server, SIP-JS [19] can run SIP in the browser, connect using WebSocket to a SIP proxy, and send peer-to-peer WebRTC media path to the other endpoint or gateway. However, this requires a SIP server that supports the WebSocket transport. We have modified this web app to run as a native desktop or mobile app, by using the native socket interface for sending SIP over UDP or TCP. Such softphones can use third-party VoIP services such as iptel.org, while keeping the media paths peer-to-peer (Fig.2f).

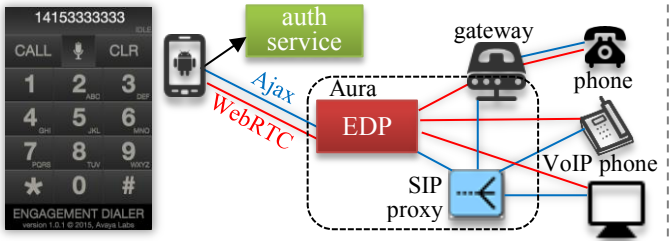
7) *LAN video phone*: This is a serverless native app (not in browser) that uses multicast to discover other app users in the local area network, and allows establishing a point-to-point video call with another user over WebRTC. It demonstrates the native socket interface including multicast that is available to native desktop and mobile apps. Unlike this, web apps must use only WebSocket or Ajax. The serverless nature of this app makes it useful for situations where set up is difficult or ad hoc interaction is desired, e.g., during emergency or in conferences.

B. *Similarities and differences*

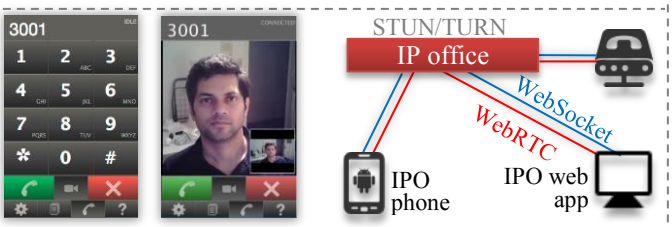
These independent apps have separate target users and requirements. Table I summarizes the high level similarities and differences in the context of this paper. To compare their relative complexities, it also shows source lines of code of combined HTML, JavaScript and CSS of the cross-platform client apps, not counting any third-party libraries or server side software. The cross-platform versions of Connected Spaces and Vclick have fewer features, and hence, are smaller than their original web app versions. Two common themes relate all these apps: (a) use of WebRTC when needed to establish a media path, and (b) use of common set of cross-platform tools to write-once run-anywhere. We present the lessons learned on these topics based on our development experience.

IV. Cross Platform Development Process

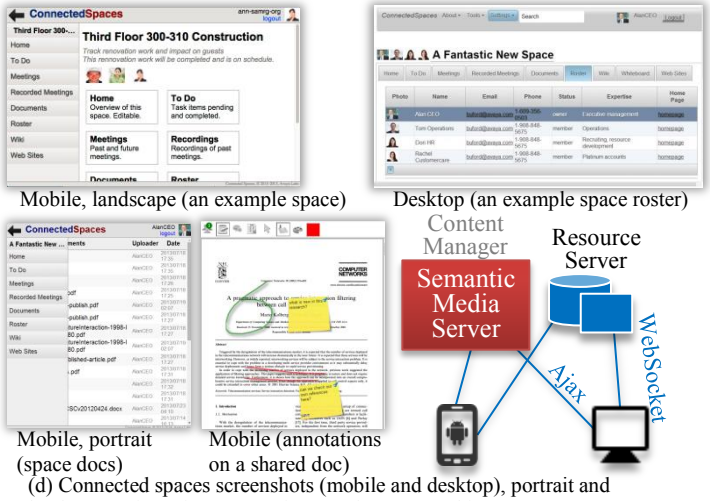
Our apps can work in a web page running inside a browser (as *web app*) on both desktop and mobile platforms, and as an installed app (*native app*) on desktop and mobile. This results in four types of cross platform scenarios shown in Table II.



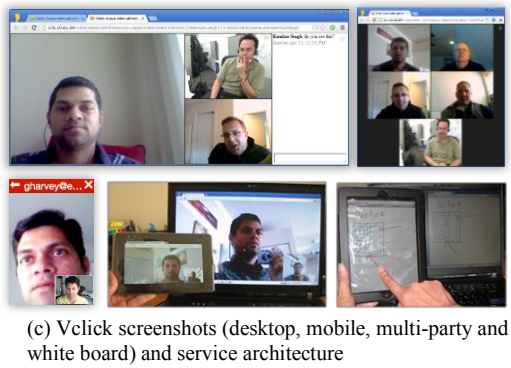
(a) Engagement Dialer screenshot and backend architecture



(b) IP office phone screenshot and IP office server

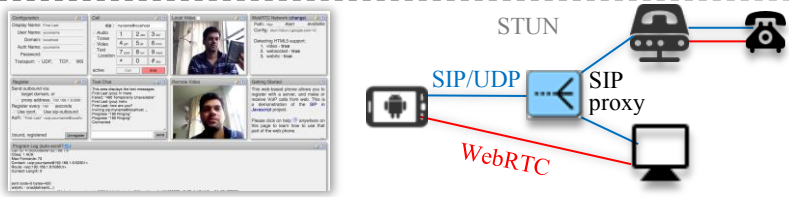


(d) Connected spaces screenshots (mobile and desktop), portrait and landscape orientation, annotations, and system architecture



(c) Vclick screenshots (desktop, mobile, multi-party and white board) and service architecture

(e) Suki app screenshots and Suki meeting helper system



(f) SIP-in-JavaScript over UDP: screenshot and architecture

(g) Serverless LAN video phone screenshots and architecture

Fig. 2. Screenshots and client-server architectures of various apps discussed in this paper

TABLE I. ATTIBUTES OF OUR CROSS PLATFORM APPS

Cross platform Apps and their requirements	Engmt Dialr	Vclick	IP off. ph.	Conn Spaces	Suki app	SIP-JS app	LAN vid.ph
Work as a web app in a browser?	✓	✓	✓	✓	✓	✓	×
Work as a native mobile or desktop app?	✓	✓	✓	✓	✓	✓	✓
Use our cloud hosted service?	✓	✓	✓	✓	✓	×	×
Active/*a*, passive/*p* or both/*ap* modes?	*a*	*a*	*a*	*ap*	*p*	*a*	*a*
Use WebRTC for media path?	✓	✓	✓	✓	×	✓	✓
Use WebSocket for signaling path?	×	✓	✓	✓	×	✓	×
Use Ajax for signaling or control?	✓	×	×	✓	✓	×	×
Use native socket interface for control?	×	×	×	×	×	✓	✓
Use client certificate?	×	×	×	✓	✓	×	×
Have real-time audio?	✓	✓	✓	✓	×	✓	✓
Have real-time video?	×	✓	✓	✓	×	✓	✓
Load or save files to/from the local disk?	×	✓	×	✓	×	×	×
Use local storage for configuration?	×	✓	✓	✓	✓	×	✓
Capture image or record audio/video?	×	✓	×	×	×	×	×
Have user notifications (e.g., call invite)?	×	✓	✓	×	×	✓	✓
Use iframe-based components?	✓	✓	×	✓	✓	×	×
Use touch interface (e.g., drawing)?	×	✓	×	✓	×	×	×
Register protocol handler (e.g., tel:)?	✓	✓	×	×	×	✓	×
Requires keep-alives with service?	×	✓	✓	×	×	✓	×
App logic is in client/*c* or server/*s*?	*s*	*c*	*s*	*cs*	*s*	*cs*	*c*
Layout fixed/*f*, stretch/*s*, zoom/*z*, adjust/*a*	*z*	*az*	*z*	*as*	*as*	*fs*	*as*
Source lines of code (x1000, kilo-)	*2.6*	*5.3*	*1.6*	*9.9*	*2.6*	*5.6*	*0.8*

TABLE II. FEATURES/CONSTRAINTS OF PLATFORM SCENARIOS

	Web		Native	
Features/Constraints	🖵	📱	🖵	📱
Requires Chrome	×	×	✓	×
Blocking alert	✓	✓	×	×
Inline scripts	✓	✓	×	×
HTML5 localStorage	✓	✓	×	×
Local files' access	×	×	✓	✓
Client certificate	✓	✓	✓	×✓
Screen share	×	×	✓	×
Built-in file dialogs	✓	✓	✓	×
Media recorder	×	×	✓	✓
Useful Ajax CORS	✓	✓	✓×	×
Useful keep-alives	✓	✓×	✓	×
Native socket intf.	×	×	✓	✓

We want to simplify the cross platform development, and highlight cross-platform differences. We use the Chrome App and Apache Cordova frameworks and tools [20][1] to write the app once in HTML, JavaScript and CSS, and run in the four cross platform scenarios. Although some apps can run on other browsers, e.g., Firefox or Safari, and other platforms, e.g., iOS, we will only discuss the use of one browser, Google Chrome, and one mobile platform, Android, in this paper. The desktop platform can be Windows, Mac OS X or Chromebooks.

Creating a cross-platform app is a three step process: (1) create the web app using HTML5 technologies following our cross platform guidelines, (2) convert it to a ChromeApp that runs and behaves as a native app on desktops, and (3) convert it to a native app, e.g., Android .apk, using Apache Cordova [20].

For WebRTC-base app development we start with a local development environment to create and test the web pages loaded from a localhost web server in the different tabs of the same browser. Once login or configuration is implemented using browser's local storage, separate browser user profiles are used on the same machine. Next, loading the web app from different machines in the same network requires opening up the firewall. As we progress towards more distributed and/or restricted network, such as Internet or across VPNs, external STUN and TURN servers are incorporated for media. Once the point-to-point media flow works, additional features such as multi-party follow the same process – test first on multiple tabs of the same browser, and incrementally spread across networks. We present our lessons learned and guidelines in creating cross platform HTML5 web apps in Section VI.

Next, we convert the web app to a ChromeApp [20]. It is a package of HTML, JavaScript and CSS files that runs as native app, but uses the browser's rendering engine and its Native Client plugin. It can be distributed on the Chrome Web Store. It runs as a standalone app and unlike a browser extension, cannot interact with or modify the visited web pages. Such apps and extensions are the only ways to install an app on Google Chromebook currently. Software in a ChromeApp differs from web pages: it can use browser specific JavaScript APIs to access local storage, file system, attached USB, power settings or raw socket interface; but cannot use certain features, e.g., document.cookie, blocking alert or prompt, form submits, HTML5 localStorage or history, inline scripts, or images and resources from external sites – all must be packaged in.

To convert, we change, among other things, the web files to move all inline scripts to a separate script file, e.g., use addEventListener instead of inline onclick; change form submit or external image load to Ajax instead of setting <img> src; and replace any restricted API to its alternative, e.g., create UI instead of built-in alert. To load third-party shared websites in an iframe, an intermediate sandboxed iframe can be used, which isolates the content security of the third-party page from the app. The alternative of webview, instead of iframe, is not portable to Cordova yet. Plugins such as Flash Player or PDF are not readily available in mobile app. In Connected Spaces, we open third-party content in a new browser tab for viewer interactions. The app also has a manifest file for its metadata, e.g., app permissions to capture audio, video or desktop, show notifications, or use cloud messaging, full screen, device's location, client or server storage, or detect when the app is idle, i.e., not receiving user focus. The manifest file also contains target URL patterns to which Ajax is allowed from this app.

In the last step, Cordova Chrome Apps (cca) tools are used to compile a packaged app to a native mobile app for the target platform, e.g., Android and iOS. A packaged app usually converts seamlessly to mobile app, e.g., app permissions are converted to appropriate features and plugins of the Android manifest file. Some features such as for file transfer or account contacts require manually adding those plugins. We selectively enable some or all of these plugins: webintent, audiotoggle, contacts, file, file-transfer, geolocation, media-capture, audiocapture, gcm, idle, notifications, storage and videocapture. We describe specific lessons learned in cross platform app development across browser, desktop and mobile in Sections VII and VIII.

WebSocket on Android is supported without a plugin. However, developers should disable "Don't keep activities" under developer options to keep the connection active when the app goes to background. The Crosswalk webview [21] used in Cordova includes support for WebRTC on Android. The camera and its auto-focus features must be manually changed

to optional in Android manifest and plugins config so that the app works on tablet devices with only user facing camera, but no rear facing. Instead of using the built-in and shared Android webview that lacks WebRTC on older versions, packing the external Crosswalk webview with each app increases the apk size by about 20MB. The tool can launch the app in an emulator or on the connected device. Emulator is slow, and causes some plugins to misbehave, so we use a connected device during development, and the host's browser to debug the target app. Once ready, the generated apk files for arm and x86 should both be uploaded to the Google Play Store.

Cordova [1] can create apps for other platforms, e.g., iOS, Blackberry, or Windows Phone, but cca [20] only supports Android and iOS yet. WebRTC is not natively available for iOS in Cordova. Third-party plugins, e.g., cordova-plugin-iosrtc and -webrtc, are emerging, and aim to provide standard WebRTC APIs. The <video> element of Safari webkit cannot be modified; hence they create an overlay for display or control of WebRTC video. Using DOM observers, the overlay adjusts to the changes in the underlying element. Since, the overlay is on top of everything else, it is not trivial to draw on it, or do CSS z-order or crop, or handle click on the video element.

V. WebRTC on a Video-enabled Wearable

Using the Glass Development Kit (GDK), one can create apps for the Google Glass, a head mounted wearable device that includes a camera and view finder. This type of wearable provides a *see-what-I-see* communication experience, and has applications in healthcare, manufacturing and energy [22][23]. The viewfinder is a low-resolution ocular display that can be controlled with voice commands as well as by touching and swiping the touchpad on the side of the glass. Due to the nature of the glass UI, the apps described in section III are too complex for presentation on the viewfinder. However, the glass is an important type of video endpoint for a participant in the team apps. Consequently, we built and tested a glass app to connect as a WebRTC endpoint (Fig.3). We used Google Glass Explorer Edition, version XE22. The app is written in HTML5 using Vclick's video component, and an apk is generated using the same process and tools described earlier. Reducing the capture dimension and frame rate reduces the video lag and improves the user experience.

VI. Guidelines for Cross Platform HTML5

1) Avoid SPA frameworks: We use JavaScript without external frameworks, e.g., jQuery or AngularJS, that are often designed for single page applications (SPA) [24] unsuitable for long lived mobile apps because they cause bulky script injection or memory leak buildup over time.

2) Componentize the user interface: Separating source files into components helps testing and maintenance. Iframes are the most widely-deployed form of web componentization [25]. Firstly, iframes to host separate components prevents leaking document, scripts or styles of one component into another. Each video box or text chat in Vclick (Fig.2c) runs in an iframe, with its own URL, and can be independently launched, e.g., to move a participant video in new tab or to open the conversation app from Connected Spaces. Secondly, the browser implicitly cleans up any residual state of an iframe component on unload. In Connected Spaces and Suki, the individual pages are loaded and unloaded on menu navigation, while still using a shared data model. Engagement Dialer loads a headless JavaScript library in an iframe, so that any per call dangling references are cleaned up when the call ends with no long term effect. On iOS, cordova-plugin-iosrtc must be invoked in the top window, not an iframe. To solve this, we proxy the API calls (Fig.5).

3) Select the right UX layout: One rigid layout does not work well cross-platform, and we optimize used space via fixed, stretched, zoomed and/or adjusting layouts. Connected Spaces (Fig.2d) and Suki adjust in landscape or portrait orientation with persistent or auto-hide menu, respectively. Vclick (Fig.2c) adjusts and zooms-in video boxes to reduce the empty space and/or to maximize the presenter's video. IP office phone and Engagement dialer are locked to portrait mode in native apps, and zoomed-in to fit the display size. No single UX can satisfy all users. In Connected Spaces we use customizable styles, so that a user can pick. Both themes in Fig.4 have same HTML and JavaScript, and differ only in CSS.

5) Use CSS for animation: Animation using CSS transition and key-frames styles is powerful, versatile and fast, compared to JavaScript. Vclick animates size and position of videos when participants join or leave. Connected Spaces and IP office glide the menu or tab on user interaction. We use the opacity style for a fade-in-out effect to indicate background activity so that the user interface contr ols can remain active and clickable.

6) Prefer intuition over instructions: Intuitive design with continuous user feedback is important [24]. Connected Spaces allows in-line edit of user profile via contentEditable instead of a separate form. IP office phone animates the answer and decline buttons on an incoming call. Engagement Dialer and IP Office Phone always show the current call state in the app.

7) Select the programming model: JavaScript is asynchronous, with exceptions, e.g., alert, localStorage. We use asynchronous wrapper around these, e.g., to change to server side storage in future or to move to native app where storage is via platform specific asynchronous function. There are many ways to do asynchronous

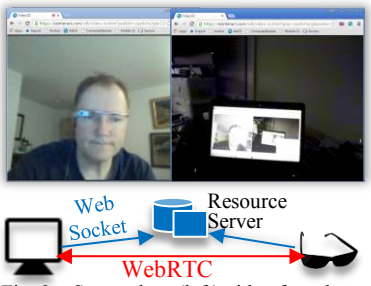


Fig. 3. Screenshot: (left) video from laptop camera to Google Glass viewfinder and (right) from Glass camera to laptop; and architecture.

Fig. 4. Screenshots of Connected Spaces themes.

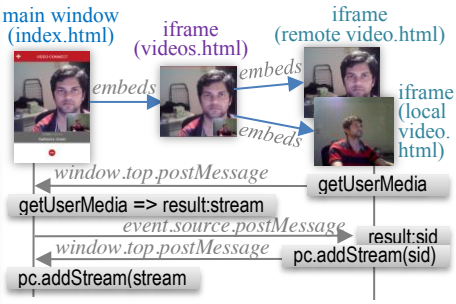


Fig. 5. Proxy WebRTC APIs from embedded iframes to the top-level window in an iOS app.

JavaScript [26]. We use callbacks for APIs and libraries, and combination of events, messaging and state machines when creating standalone components.

VII. Lessons Learned in Cross Platform WebRTC

1) Media capture and playback: Unlike tablets or laptops, smart phones have two cameras (front, rear) and two speakers (speakerphone, earpiece). In a multimedia call, user should be able to select the right device, e.g., earpiece in audio call and speakerphone in video. If an incoming native phone call corrupts the audio settings in the app, it should revert to user defined ones when the app regains focus. Mobile browsers (not apps) restrict automatic playback of media files loaded from web without user click, causing problem in ringer sound. For mobile web apps, we pre-load the media file or embed the media content as data URL in the source code. To avoid local sound loopback, we mute the <video> tag to which WebRTC local media stream is attached for preview. This also applies when WebRTC and Web Audio APIs are used to record voice by attaching a local media stream containing microphone to an audio context. Full screen mode to maximize video is useful for desktop and mobile. Android camera capture is squarer in portrait and standard or HD in landscape device orientation. WebRTC allows capture size constraint, but not on active local stream. Differences in capture and display size are handled by zooming in to the center of the video (see Fig.2b/c/f/g).

2) Network connectivity: Selecting the right network interface saves cost and improves call quality. Cost or security of Wifi vs. 4G vs. VPN is not used in WebRTC address selection yet. However, an app can ask the user to pick the right interface, and apply that to the onicecandidate callbacks. An app can also accommodate network change, e.g., by reconnect of signaling or media path. Low bandwidth constraints on mobile devices can be used to negotiate low framerate or capture dimension in a call. A STUN server is enough for most residential users. A cloud hosted TURN server often achieves relayed media path for restricted enterprises and VPNs. More restricted enterprises require opening up firewall ports to such TURN servers.

3) Interoperability: Our apps can call between web and native versions. Different WebRTC code in browser vs. Crosswalk webview or plugin can sometimes break interoperability, e.g., when Google Chrome added the useinbandfec=1 SDP attribute which was not understood by webview. We solved this by intercepting WebRTC APIs and removing this attribute. Cross app interoperability or with server-side WebRTC, e.g., in IP office phone and Engagement Dialer, is a different problem, and is often caused by differences in WebRTC stacks including ICE and STUN/TURN. WebRTC refines many SDP attributes [4][19] breaking interop with existing VoIP systems.

4) Multi-way call: There are three ways to do multi-party: full mesh (Vclick) is like multiple two-way calls [17]; centralized mixing uses client-server media with server side video compositing or switching; and server forwarding uses one upstream and N-1 downstream media paths. System scalability is governed by the number and bandwidth of media streams at the endpoint or server. Full mesh congests the user's uplink with only a few participants. Other ways have server scalability issues. Native apps on desktop or mobile can use native socket interface to realize a fourth way of efficient multicast, at least on local network. Our LAN video phone uses multicast for discovery. However, using multicast for media requires using a custom media stack instead of peer-to-peer WebRTC flows.

VIII. Other Cross Platform Development Techniques

1) User input: On mobile, the HTML <input> tag with type of microphone, camcorder or camera can launch the native capture application. There is no built-in file-selector or save-as dialog in Cordova, and the file plugin's API is incompatible with HTML5's. We built own UI to save text chat or to select a file to share in mobile Vclick app. Connected Spaces and white-board of Vclick allow a user to draw via touch input. The keypress event instead of keyup/down works better with virtual keyboards of mobile. The back buttons available on some Android devices trigger the in-app back button behaviors in Connected Spaces and Suki.

2) Cross origin (CORS): A ChromeApp has origin of chrome-extension://{ext-id} and a Cordova window or iframe has origin of file://, gopher:// or chrome-extension:// (no path). A website that enables white-listed origins should consider such wild-card values. We use nginx to terminate request, do out-of-band authentication, and then proxy to the website. Same origin components of different iframes can do seamless function and variable access; or use postMessage for cross origin.

3) App in endpoint: WebRTC is a client-side technology. Many of our apps run most of the app logic in the client's JavaScript. This promotes scalability, robustness and reusability. Client apps such as Vclick widgets are reused without complex server side interactions. We use HTML5 registerProtocolHandler and Android web-intent to launch an app when the browser opens, say, a "tel:" URL. A known issue in Chrome prevents auto-loading of external app without a page refresh in JavaScript. Opening web+vclick:*user@domain* dials-out, and web+vclick:call:*call-id* joins the call via Vclick. Connected Spaces can launch Vclick with call-id as MD5(doc-url:spacename) to isolate the interaction on a document of one space from that on the same document shared in another. Such loosely coupled interactions enable app innovations, e.g., a new app can replace Vclick without changing Connected Spaces, or a "facetime:" URL can reach Vclick from an iOS device via a gateway in the future.

IX. Real-time Notifications: Power and Security

A. Power management with notification channel

Some apps may work offline, but others require network connectivity. Persistent WebSocket channel is useful on web pages and desktop apps to receive asynchronous events such as call invite. Periodic keep-alives keep the connection active. When it fails, reconnection is attempted. Vclick does automatic failover to the secondary server when the primary fails.

To conserve power, long running native mobile apps should not keep persistent WebSocket connection. Keep-alives or unbounded reconnection attempts are resource intensive, and should be reduced, e.g., Vclick Android app stops the keep-alives when device goes to sleep or is not in foreground, uses exponential back-off timer with a cap to attempt reconnections, and stops them if the failure persists for some time. When the device becomes active again, it does a one-time check to detect connectivity and to reconnect if needed.

Without persistent WebSocket connection on native mobile apps, user presence and app's ability to receive asynchronous events are not trivial. Vclick uses low power shared Google Cloud Messaging (GCM) that can receive events even when the device is asleep. When received, it then wakes up if needed, and connects to the server to fetch further call data. The caller publishes the outgoing call event to both the server and GCM, in case the receiver device is asleep or not connected to the server. Similarly, the server should inform all the devices after recovery from a failure, so that the devices that had stopped their reconnection attempts can now connect.

To receive network events during sleep, an app should either use cloud messaging or, on native mobile app, mark the native socket as persistent, so that it is kept active even when the app is unloaded, and can be reused when the app is loaded back. Other platforms have alternatives to GCM, and are particularly useful on iOS where the browser terminates WebSocket when the device goes to sleep.

B. Security: cloud connectivity and hosting

The challenges and techniques in creating our multi-tenant cloud portal are described in [3]. We use combination of secure TLS transport, client certificate, authentication, cross-origin restrictions and nginx to securely access the cloud services. Client certificates work on web apps and native desktop apps. For native mobile apps, support in Android and Crosswalk webview was recently added. We created a Cordova plugin to set the client certificate from web page. It works for Ajax requests. We discovered other issues: the default download manager does not honor client certificate; WebSocket fails if the server requests a client certificate, unless a previous Ajax to the same server caused the certificate to be cached. We use nginx to provide an alternate path from native mobile apps to such services; the app connects to the proxy without client certificate but with an auth-token, and the proxy connects to the cloud service with the correct client certificate.

X. Conclusions and Future Work

We have described seven different cross platform apps built using ChromeApp and Apache Cordova frameworks and tools. These apps use WebRTC for real-time streaming audio and video, and typically interact with cloud servers on HTTPS. The code is written in HTML5 and is portable and interoperable across in-browser, desktop and mobile app versions. Platform specific differences related to collaboration apps are discussed. We have developed many possible collaboration scenarios, and have presented challenges and techniques to solve some difficult problems to help other cross-platform developers.

We are porting our apps to other platforms (iOS), using an external WebRTC plugin in Cordova. We plan to re-create one app using multiple hybrid frameworks to compare them against Cordova. Impact of software based VP8 codec vs. native H.264 on battery consumption is for further study.

Acknowledgments

Thiru Arjunan, Jaydeep Bhalerao, Biswajyoti Pal and Ajita John helped in getting started with Avaya IP Office, EDP and Suki. Venkatesh Krishnaswamy supported our research and development.

References

[1] Apache Cordova, https://cordova.apache.org/, retrieved Jul 2015.

[2] ALICE: Avaya Labs Innovations Cloud Engagement, https://alice.avayalabs.com, retrieved Jul 2015.

[3] J. Buford, K. Singh, and V. Krishnaswamy, "ALICE: Avaya Labs Innovations Cloud Engagement," Principles, Systems and Applications of IP Telecommunications (IPTcomm), Chicago, IL, USA, Oct 2015.

[4] A. Johnston and D. Burnett, WebRTC: APIs and Protocols of the Real-Time Web, third edition, Digital Codex, 2014, ISBN 978-0985978860

[5] The WebSocket API, W3C candidate recommendation, Sep 2012, http://www.w3.org/TR/websockets/

[6] K. Singh and V. Krishnaswamy, "Building communicating web applications leveraging endpoints and cloud resource service," *IEEE Intl Conf on cloud computing* (IEEE Cloud), Santa Clara, CA, USA, 2013.

[7] K. Bell, http://mashable.com/2015/01/15/google-play-more-apps-than-ios, Web page, retrieved Jul 2015.

[8] J. Raj, 2014, http://www.sitepoint.com/top-7-hybrid-mobile-app-frameworks, Web page, retreived Jul 2015.

[9] I. Malavolta et al. "End Users' Perception of Hybrid Mobile Apps in the Google Play Store." 4th IEEE International Conference on Mobile Services, New York, USA, Jun-Jul 2015.

[10] H. Heitkötter, S. Hanschke, and T. A. Majchrzak. "Evaluating cross-platform development approaches for mobile applications," *Web information systems and technologies*, Springer Berlin Heidelberg, 2013, pp.120-138.

[11] E. Angulo and X. Ferre, "A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX", in *Proceedings of the XV International Conference on Human Computer Interaction* (Interacción '14), ACM, New York, NY, USA, 2014,

[12] S. Xanthopoulos and S. Xinogalos. "A comparative analysis of cross-platform development approaches for mobile applications," *Proceedings of the 6th Balkan Conference in Informatics*, ACM, 2013.

[13] N. J. Pierre et al., "Cross-Platform Mobile Geolocation Applications Based on PhoneGap," Lecture Notes on Soft. Engg. 3.2 (2015): 78.

[14] S. Amatya and A. Kurti, "Cross-platform mobile development: challenges and opportunities," ICT Innovations 2013, Springer International Publishing, 2014, pp.219-229.

[15] R. Kazi, X. Zhang and R. Deters, "Supporting apps in the personal cloud: using WebSockets within hybrid apps." Second Symposium on. Network Cloud Computing and Applications (NCCA), 2012.

[16] S. R. Humayoun, S. Ehrhart, and A. Ebert, "Developing mobile apps using cross-platform frameworks: a case study." Human-Computer Interaction. Human-Centred Design Approaches, Methods, Tools, and Environments, Springer Berlin Heidelberg, 2013, pp.371-380.

[17] K. Singh and J. Yoakum, "Vclick: endpoint driven enterprise WebRTC", (to appear in) IEEE International Symposium on Multimedia (IEEE ISM), Miami, FL, USA, Dec 2015.

[18] J. Buford, K. Mahajan and V. Krishnaswamy, "Federated Enterprise and Cloud-based Collaboration Services," *IEEE Intl Conf on multimedia system architectures and applications*, IMSAA, Bangalore, India, 2011.

[19] K. Singh and V. Krishnaswamy, "A case for SIP in JavaScript", IEEE communications magazine, Vol. 51, No. 4, April 2013.

[20] Run Chrome Apps on mobile using Apache Cordova, https://developer.chrome.com/apps/chrome_apps_on_mobile, Jul 2015.

[21] Crosswalk project, https://crosswalk-project.org, retrieved Jul 2015.

[22] Glass at Work, https://developers.google.com/glass/distribute/glass-at-work, retrieved Jul, 2015.

[23] A. Barr, "Google Quietly Distributes New Version of Glass Aimed at Workplaces", The Wall Street Journal, Jul 30, 2015.

[24] S. Porto, 2014, Lessons learnt by building single page applications, https://reinteractive.net/posts/186, retrieved Jul 2015.

[25] T. Leithead and A. Eicholz, "Bringing componentization to the web", http://windowsforum.com/threads/211241, retrieved Jul 2015.

[26] Online article, http://tech.pro/blog/1402/five-patterns-to-help-you-tame-asynchronous-javascript, accessed Jul 2015.