Hello everyone. My name is John Buford.

This paper is about cross-platform WebRTC team apps. Particularly, how to write once and run anywhere for WebRTC apps? What are the challenges of creating cross-platform team apps? What are the lessons learned from our practical experience with several apps? And what are the guidelines we share to help other developers?

Kundan Singh is my co-author.

The focus is on three things: cross-platform, WebRTC and team apps. Our team apps deal with communication and collaboration use cases, particularly for enterprises. And we use Apache Cordova and Chrome Apps as cross-platform development tools.

The paper shows how to realize write-once-run-anywhere for such apps, and what are important lessons learned from our experience.

Our work focusses on communication and collaboration apps, and presents the collective knowledge from our experience developing several apps.

In particular, Fig.2 shows the seven different apps seen in enterprises and covered in the paper. These are good selection of representative apps and services found in enterprise team collaboration.

The first one is engagement dialer, which is basically a phone number dialer using our VoIP servers and gateways.

The second is a voice and video client for our IP office product for small and medium businesses. The client app connects to the cloud server.

The third one called Vclick is basically as collection of many endpoint driven independent apps, that use a light weight server in the cloud, and run all the app logic in the endpoint. It has video call, conferencing, presence and various forms of collaboration such as screen sharing, shared notepad and whiteboard.

Unlike previous IP office system with client-server media flow, Vclick uses peer-to-peer media flow.

Connected spaces is a team collaboration system for persistent sharing of content such as documents, websites, meeting notes, etc., within organized team spaces.

It also allows annotations and impromptu conversation with voice, video, text in the context of a shared document. The server runs on Amazon cloud.

Suki meeting helper agent joins your conferences and captures meta data and notes, and at the end of the meeting sends it to you. The Suki client app shows past meeting summaries in your calendar.

Unlike engagement dialer or IP office phone, that run the VoIP stack on the server, our SIP-JS client runs the SIP (or session initiation protocol) stack in the endpoint.

The benefit is that it can connect to existing VoIP providers' SIP proxies without needing specialized WebRTC gateways, and supports voice, video and text.

Finally the LAN video phone uses local area network multicast to discover other users, and can do peer-to-peer voice and video call, without using any server.

These apps have two common themes: first, all the apps except one (Suki) use WebRTC for real-time voice and video flows, and second, all the apps use the same set of cross-platform tools to write one and run on wide range of platforms.

This shows the WebRTC media flows in various apps.

Some apps use client-server media path, and others use peer-to-peer or full mesh. The media server when used can facilitate additional features such as gatewaying or call recording.

The signaling is done using Ajax or WebSocket.

Some of the services are hosted on cloud for our trial..

The paper only deals with the client apps, not the servers. Table I summarizes the similarities and differences among these apps.

For example, two of the apps use client certificate for added authentication. Two apps deal with load or save of files to disk. Four of the apps use iframe-based components in UI implementation. And three of the apps require periodic keepalive or presence.

Each of these apps can run in range of cross platform scenarios. We particularly focus on four categories: web app within a browser on PC and mobile, and installed app on PC and mobile.

Table II summarizes the differences in constraints on these platform categories. For example HTML5 localStorage is available only for web app, not installed. Whereas native socket interface, e.g., for UDP or multicast, is available only for installed apps, but not from within a browser. Client certificate is generally supported everywhere, but on installed mobile app, the support is ongoing and not fully functional. Some details are in the paper.

We use WebRTC for audio and video flows. WebRTC is an emerging technology for plugin-free browser to browser multimedia flows.

Fig.1 shows the two high level abstractions: get-user-media is used to capture the local stream from camera and microphone, and peer-connection is used to represent peer-to-peer channel. Fig.1 shows how the media captured from one browser is attached to the peer connection and displayed on the others, in both directions.

We abstract WebRTC and related code in a high-level video widget. The video widget is used in few of our apps as well as on wearable devices like Google Glass.

The video widget is basically a video box which can publish or play a named stream using a cloud hosted resource server for signaling.

This shows different video widgets publishing and playing two streams. The actual stream can be with peer-to-peer media flow of WebRTC.

There can one publisher for a stream, and zero or more players. The publishers and players can come and go in any order, and the widget correctly establishes the media flows. This concept is used for wide range of WebRTC apps such as video call, conference, or video presence.

Let me show a quick video demonstration of this video widget running on various platform categories.

This is the demonstration of the video widget running on wide range of platforms. So I have a windows laptop, a mac book, and few mobile devices both android and iOS.

The video widget is capable of publishing or playing. By default it publishes the local video (and audio) on start up.

You can see the local video on this windows laptop, where it is running as a web application in the browser.

On the mac book, I will launch the installed application. It has the same behavior and same user interface, and publishes the local video from the mac book camera.

Now I can attach the windows video widget to the published stream of the mac book video widget. Since video widget can publish or play, now I have one publisher from mac book, and one player on windows, both showing the video stream of the mac book camera.

Next, I launch the installed mobile app of video widget on an android phone. On start up, like before, it shows the local video from the android phone camera.

Now I can change the two laptop video widgets to play this video stream from the android video widget. Here there is one publisher and two players for this video stream.

Next, I will open the web app of the video widget on android tablet device. Same behavior, it publishes local video first, and I can change it to play, say from the mac book video widget stream.

The video widget app is also installed on the iPad, i.e., iOS device. By default it publishes the iPad video. And I can change the two laptop video widgets to play this iPad video stream.

That all about the demo of this video widget abstraction of WebRTC.

So how did we create these cross platform apps for write-once-run-anywhere?

It is a three step process: first, write the HTML5 code for the web app to run in the browser. Second convert it to a chrome-app to be installed on PCs. And finally, convert it to a cordova-app to be installed on mobile devices.

A chrome-app is basically a packaged collection of web files, which run and behave like a native installed app on PCs.

Converting a web app to chrome app is not trivial due to several constraints. For example, it cannot have inline scripts, or inline load of images or other resources from external server. The paper has more details on the challenges and techniques.

Apache Cordova is a cross-platform development tool that converts web files to native installed apps, e.g., for Android and iOS. There are many plugins that accomplish individual features, e.g., for storage or media capture.

Converting from chrome app to cordova-based native app is not difficult using the chrome-cordova-apps (cca) tools. However, some features are not available in Cordova, and must be re-implemented, e.g., open or save file dialog.

In summary, these are the three steps detailed in the paper.

The paper also has guidelines on how to create HTML5.

For example, what are the problems with single-page-apps frameworks like AngularJS. And how do we use iframe-based components.

An example of componentization is shown in this connected spaces UI, where the same HTML and JS but different CSS produces entirely different user interface.

This is the connected spaces user interface in portrait and landscape mode, and layout automatically hides the menu in portrait mode.

This shows the four types of layouts. Depending on the application, often times, the stretched, zoomed and/or adjusting layout is used, instead of fixed.

For example, the Vclick app adjusts the participant video layout as the window size changes, e.g., going from portrait to landscape mode.

Using CSS instead of JavaScript for animation is fast, and works well when converted to native apps using Cordova.

For example, this shows heart-beat effect when the app is doing some background task.

And this one shows how the participants video boxes animate as people join or leave a Vclick conference.

The paper as more details on how to prefer intuition over instruction, and how to select the right programming model for designing APIs and widgets.

The paper also shows specific examples of audio/video related tricks and techniques.

For example, mobile web browsers do not load sound without user click, so how do we play incoming call alert.

Android gives squarer capture dimension in portrait mode, so how do we adjust the layout.

How can we restrict the IP addresses used by WebRTC to only VPN interfaces, when doing private office conversation?

What are the differences between centralized and peer-to-peer media flows?

Even non-WebRTC tips are presented in the paper.

For example, some HTML features are available on mobile but not on PCs and vice-versa.

Web origin of a chrome app or cordova app can be vague, what problems can it cause and how to solve those?

What are the benefits of endpoint driven apps?

How do we launch an external app on web or on installed mobile app?

Finally, there are some techniques for dealing with cloud based apps, especially for security and performance.

For example, how to save battery by not doing keepalive on WebSocket on mobile, but still be ready to receive incoming call?

Why does WebSocket fail sometimes? and how to deal with that?

How to improve transport security and access control using client certificate? And how to deal with if a client or server does not support client cert?

So that brings me towards the end of the presentation on cross-platform WebRTC team apps.

To summarize, the paper contains these topics.

Thank you for listening.