Vclick: Endpoint Driven Enterprise WebRTC

Kundan Singh and John Yoakum
Avaya Labs Research
Santa Clara, CA and Cary, NC, USA
{singh173,yoakum}@avaya.com

***Abstract*—We present a robust, scalable and secure system architecture for web-based multimedia collaboration that keeps the application logic in the endpoint browser. Vclick is a simple and easy-to-use application for video interaction, collaboration and presence using HTML5 technologies including WebRTC (Web Real Time Communication), and is independent of legacy Voice-over-IP systems. Since its conception in early 2013, it has received many positive feedbacks, undergone improvements, and has been used in many enterprise communications research projects both in the cloud and on premise, on desktop as well as mobile. The techniques used and the challenges faced are useful to other emerging WebRTC applications.**

Keywords—WebRTC, enterprise communication, web video conferencing, resource-based architecture, web applications

I. Introduction

Existing web-based video collaboration systems exhibit one or more of these issues:

Tight coupling: An application that controls the user data or is controlled by a single VoIP provider encourages vendor lock-in and prevents open innovation.

Thin client: This at a price of a thick server, with bulky, stateful or complex service logic, is hard to make scalable and robust.

Rigid user interface: A rigid video conference dashboard with boxes for videos, chats or rosters often has product limitations, e.g., only one video per user or one screen share in a meeting.

Single page applications: An app using a single page or plugin framework is hard to maintain due to monolithic code, slow to respond due to memory leaks buildup or single execution context, and slow to develop around such frameworks.

Painful first step: The time and effort to successfully make the first call are often high due to external dependency, inability to call self, or to do multiple logins for the same user.

We present *Vclick* to address these issues. Unlike a rigid piece of opaque software, it is a collection of many loosely-coupled web apps. For example, a video call can involve a call invitation via a browser extension (first app) that delivers the "invite" event and a URL to the receiver, and when clicked, it opens the video conversation page (second app) for that call. The second app does not care even if another first app is used, e.g., an email to deliver that URL. Browser is the only user tool, and the user is able to run any app – conference, chat or roster – independent of her data storage, e.g., for contacts or conversations. The apps or *widgets* in Vclick run in their own separate tabs or *iframes* (Fig.1), behave independent of each other, mash-up at the data level using a thin server, and run the entire app logic in JavaScript in the browser.

We present Vclick's distinguishing software architecture having a range of collaboration apps: audio/video conference, text chat, file, screen or app sharing, shared white-board and notepad. We use HTML5, WebSocket [1] and WebRTC (Web Real Time Communication) [2] for a pure browser based communication which is independent of VoIP systems, e.g., using SIP (Session Initiation Protocol) [3]. Section II has background and related work. Section III presents the system architecture. Section IV covers the lessons learned and impact generated. Section V has conclusions and future work.

II. Background and Related Work

WebRTC [2] enables a web page to establish a peer connection between two browsers and transport captured media from one to another without a plugin. Fig.2 shows various elements in an asymmetric call with audio and video in one direction and audio-only in the reverse. Unlike a SIP [3] phone that often limits the number of audio or video streams, the WebRTC API is more flexible. Like a SIP proxy, WebRTC needs a *notification system* to exchange certain signaling data between the browsers. We use *resource service* [4] as a shared data access and notification system. Further details are in [4] and summarized in Fig.3. Unlike SIP, we separate the three pieces of call invite/answer, session description and transport addresses to decouple the call initiation from the conversation. The resource semantics are defined by the client apps, e.g., /users/bob/presence represents a user's presence, and any change in that is propagated to all its subscribers.

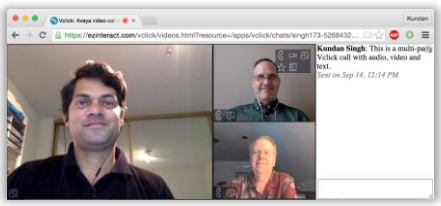

Fig.1. Screenshot of the Vclick conversation page in a 3-party call. Each video box or text chat is a separate widget with its own URL and runs in a separate iframe.

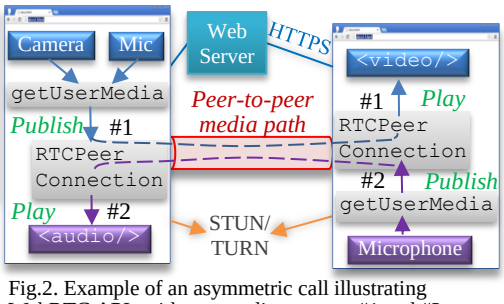

Fig.2. Example of an asymmetric call illustrating WebRTC APIs with two media streams, #1 and #2.

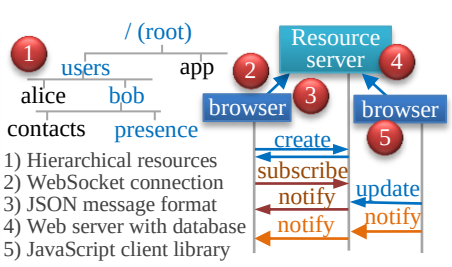

Fig.3. Resource service as a generic lightweight data access and notification system for WebRTC.

Fig.4. Screenshots of Vclick apps on mobile. (L) conversation apps with video on tablets. (R) a separate tablet is used to join the white-board app of the call where voice and video are on desktop.

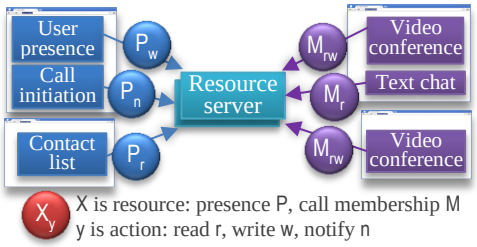

Fig.5. Example of loosely coupled apps running in browsers that mash-up at the data level, and no single app controls the shared user data.

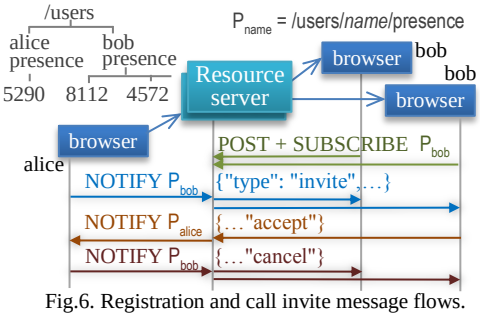

Fig.6. Registration and call invite message flows.

Numerous web apps using WebRTC and WebSocket have emerged in the last few years [5]. The novelty of our work is not in creating yet another application, but in the following:

1. Loosely coupled independent apps, unlike a single rigid app, that mash-up at the data-level and create the complete user experience (see Fig.4 and 5).
2. Separation of user data from the app logic: the same app can tie to either a public cloud or private (enterprise-only) resource server.
3. Ability to run all the app logic in the browser so that the service can easily be made robust and scalable.

The apps are useful beyond enterprise use, e.g., one author has been extensively using the video app for baby monitoring.

III. Software Architecture and Implementation

A transient app, e.g., video call, shared notepad or text chat, in Vclick is launched by its URL with the right parameters. A persistent app, e.g., presence or click-to-call, needs a browser extension or a native app and runs behind the scenes.

A. *Execution context: browser extension, web page, iframe*

Vclick's browser extension gives an execution context to the persistent apps. The presence app marks the user as available to receive calls if her browser is open. Click-to-call app modifies any visited web page to inject click-to-call icons next to phone numbers. The extension [6] runs in Google Chrome, but is not available on mobile platforms. It shows an icon next to the browser address bar to initiate a video call. An incoming call creates desktop notification to answer or decline.

When the call initiation app initiates or answers a call, the extension opens and keeps track of conversation pages. Individual transient apps use iframes as execution contexts, including nested ones for participant's video apps. Using iframes creates a modular design, promotes loose coupling (Fig.5), and avoids problems of single page apps. A single video can be drag-and-dropped from the conversation page to a separate browser tab, to run a copy of the video app in that tab to display only that participant, independent of the ongoing call. Also, new apps are added without changing existing ones, e.g., contact list and meeting scheduler are just separate web pages.

B. *Guidelines for presence, call initiation and conversation*

The apps in Vclick (1) do not mix call initiation with conversation features, (2) keep all the app logic in the browser, and (3) allow multiple registrations from the same user. The first point allows a user to join the conversation irrespective of how she obtains its URL, and keeps call initiation independent of conversation media or widgets – audio, video, text, white-board and notepad. It also affects some design decisions: call control progress or reject reason data are kept inside the call initiation app instead of leaking to the conversation page.

The second principle enables data level mash-up instead of pair-wise app-specific integrations, e.g., another user approved third-party app can publish her presence, or she could keep video and text on her desktop but use her tablet as white-board (Fig.4) or another camera source without complex call control. An app directly accesses the necessary resources at the server, e.g. the conference app monitors the members list to handle join or leave, and to add or remove the nested video apps.

Finally, the third principle enables user login from multiple browsers or devices, and calling self for testing. User's email address is her identity. Fig.6 shows the presence apps of two browsers creating and subscribing to two child resources under user bob. Subsequent call messages such as invite, accept, reject or cancel on the target resource, /users/bob/presence, are delivered to all the subscribed apps. Once the call is answered or declined, a cancel is sent to clear the pending invitation on other devices. Unlike INVITE and REGISTER semantics in SIP, we use generic data access and notification. It extends to, say, multiple call memberships from a user, or many conversation apps. Unlike a SIP proxy that sends cancel, we let the caller do that, to move the transaction state to the endpoint, and to avoid the one-off INVITE with CANCEL semantics at the server.

C. *Login and call initiation message flows*

In Fig.7, the login state indicates if the app has connected to the server and posted the presence resource. Clicking the extension icon (outbound invite) or receiving an incoming invite creates a new invitation state. When the invitation is accepted, declined, timed out or cancelled, the state disappears. Messages in the same invitation have the same invite-id attributes. For outbound call, the app creates a new conference resource and sends its path in an invite to the target. When it is delivered to one or more targets, the state changes from wait to inviting. For incoming invite, a desktop notification is shown, when clicked sends an accept back to the inviter. The failure cases are in grey arrows, e.g., if the invite is not delivered in wait, or the caller gets a reject or closes the conversation page in inviting, then the

Fig.7. Login (presence) and call initiation state machines in Vclick.

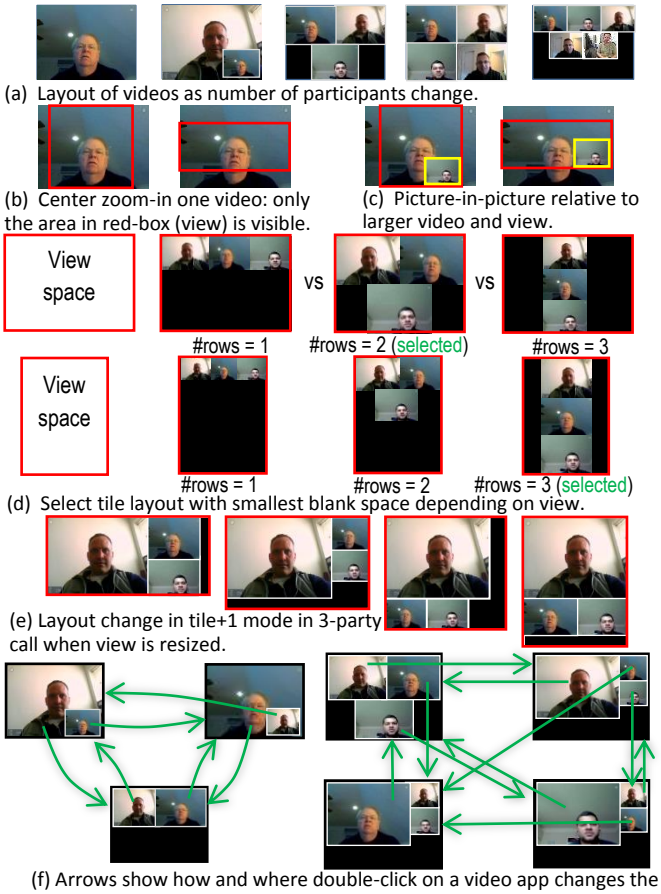

Fig.8. Layout algorithm of video app iframes within a video conference app.

state transitions to done, and it optionally sends cancel and/or shows error. If the receiver closes the incoming notification, does not respond in time, or gets a cancel, then it transitions to done, and optionally sends reject and/or shows a missed call notification, which on click calls back. The conversation app is launched in a new browser tab when the state changes to inviting or from invited to done. Call disconnection is handled in the conversation apps, freeing the initiation app. The extension icon can be used to invite more people to make it multiparty.

D. Multiparty multimedia conversation apps

The conversation app gets the URL parameters for the conference resource path, local member's identifier and name, which are used by the embedded video and text widgets to join the call. All membership and message resources are created under the conference resource, e.g., /app/vclick/chats/16352. The local member-id has a random suffix, e.g., bob-726, so that a user can join multiple times in the same call. A member resource has media attributes, e.g., mute status. Unlike session re-negotiation in SIP, we use shared object attributes for call hold or participant mute. An app subscribes to the conference and its member resources to detect when something changes.

The conversation page shows the video conference app, and optionally, the text chat. These apps appear side-by-side or one-below-another in landscape or portrait orientation. The text chat stores the messages under the conference resource instead of using a separate chat protocol. It supports emoticons, click-able link detection, and file sharing using client-side app logic,

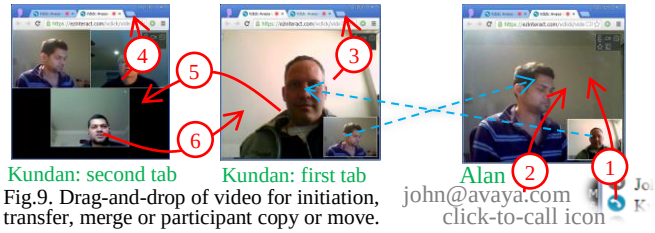

Fig.9. Drag-and-drop of video for initiation, transfer, merge or participant copy or move.

e.g., file drag-and-drop with data URL in the sent messages and clickable blob URL in the displayed messages. Other apps (notepad, white-board or screen share) are opened in new tabs.

The conference app has video apps, one per participant. Our algorithm automatically adjusts layout in each client when the available browser size or videos count changes, or the user double clicks to alter the layout mode. It shows picture-in-picture for two-party, and tile for multiparty. It minimizes the blank space to determine the video sizes and positions. Multi-party can be in tile of same size, or tile+1, with one presenter video larger than others. Fig.8 shows the algorithm and, with drag-and-drop, double-click and zoom capabilities, supports a range of scenarios, e.g., presenter on separate monitor, device orientation on tablets, or screen share from another device.

Fig.9 shows call initiation, transfer or merge via drag-and-drop to invite to an existing call (1, 2); transfer a member to a new call in a new tab (3); create sidebar call with a person in a new tab (4); merge (move) a two-party call to another (5); or invite (copy) a member from one call to another (6); The copy-vs-move semantic (5, 6) is non-trivial and needs careful design.

A video app represents a participant; with one publisher and N-1 players in an N-party call. We use unidirectional media flow in a peer connection; with another publisher-player in reverse. This allows simple app abstraction to implement two-party, multiparty, one-way and broadcast scenarios. A publisher or player can come or go in any unsynchronized order. The app correctly creates connections and media flows without custom call signaling, e.g., if a video is moved to a separate tab or call. Further video widget interactions are detailed in [4].

IV. Lessons Learned and Impact Generated

A. Scalability: signaling and media paths

We use client-server signaling and full mesh media paths. The resource service scales like a pub/sub system using data partitioning, in-memory cache and subscriber aggregators. A read or write accesses the database, whereas notify may not. The browser implicitly mixes audio of apps. Local bandwidth is enough for small conferences. CPU quickly gets saturated with a 3-party call, and affects quality after that. CPU usage depends on the video display size (player), and the streams count (publisher). In the future, WebRTC APIs will allow adjusting quality and bandwidth. Beyond 3-party, we can switch to a centralized media server with one send/receive stream per client, further reducing the upstream bandwidth and publisher's CPU usage. Here, active speaker switches or tiles videos. A hybrid conference for VoIP interworking with mix of full mesh and bridged participants is for further study.

B. Interoperation with telephony gear

Our extension can also inject click-to-call icons on phone numbers to initiate phone calls via a WebRTC-to-SIP gateway.

A multiparty call with mix of browsers and phone users is for further study. A web app to access a telephony-centric system is easy, but fully including a telephony endpoint in a pure web-centric system such as Vclick is not, due to these differences: full mesh vs. centralized, uni/bi-directional flows, or missing trickle ICE on existing gateways. Gateways often impose artificial restrictions, e.g., at most one peer connection or one media stream in a peer connection, or inability to change active stream. The endpoint driven call control does not integrate easily with session-based telephony where peer-to-peer flows must be in a call state. We plan to use server side widgets to interoperate and interact with both the resource server and the gateway.

C. Mobile devices and other browsers

On mobile devices or other browsers without our extension, the presence and call initiation apps can run on a web page (Fig.4). The user can receive or make calls if that page is open. Other issues of running Vclick in a mobile browser follows:
1) Mobile browsers disallow loading a media file (e.g., ringing) without user action (or click). We pre-load the file on startup, and play it on call invite. Alternatively, data URL can work. The audio element's loop attribute does not work.
2) Android devices give square camera capture ratio in portrait mode instead of standard 4:3. The videoWidth/Height of the video element allows zooming-in to avoid black paddings on sides.
3) The touch interface is handled separately and sometimes differently than mouse, e.g., to drag picture-in-picture video.
4) WebSocket behavior on device sleep depends on platform. Android keeps it alive for some time, and can receive incoming call, but iOS disconnects it, making it useless for presence.

We have also built a native Android app using the Chrome Cordova App framework. It uses Google Cloud Messaging (GCM) for incoming call when the device is asleep. We use cordova-plugin-iosrtc on iOS; which does not work in iframes, so we use postMessage to proxy API calls to the top-level window.

D. Private enterprise-only vs. public cloud deployment

We have done many deployments including enterprise-only (on premise) and public (on cloud). Media path is peer-to-peer on premise for privacy, but may be relayed on cloud. The cloud resource server has improved security, resource-level access control, connection-level authentication transparent to the user, and per-device customization for the same user. WebSocket and WebRTC sometimes fail on cloud due to enterprise web proxies, poor connectivity or server failure. Interference in WebSocket handshake and its disconnection by old-style web proxies are solved by careful configuration, use of TLS and/or reconnection. Our apps reconnect to the resource server using a randomized exponential back-off timer with a cap, and failover to a secondary server when the primary one fails. The client and server both have keep-alives at different layers. The peer connection is reattempted if it breaks in a call. Finally, nginx reverse proxy terminates TLS, e.g., to verify client certificates.

E. Use in our other research and development projects

Our team collaboration system uses Vclick apps to allow impromptu interaction among viewers of a shared document, and to annotate. The resource server is available in Python, PHP and Java, and uses Postgresql, MySQL or an in-memory database, respectively. For Avaya Engagement Development Platform, we modified the apps to use SockJs, and a single multiplexed connection in the top level web page to reduce the socket overhead of Ajax. Our Python server is less than 2k lines of code on top of Google's WebSocket implementation. It can run on a single virtual machine or laptop. We have reused some of the Vclick apps with our Avaya's IP office and media server. Our video presence app uses Vclick, publishes periodic snapshots of the page visitors and allows one-click to initiate or join a call in a virtual office scenario.

V. Conclusions and Future Work

Vclick is a web video collaboration system using HTML5 web technologies including WebSocket, WebRTC, drag-and-drop, and data/blob URLs. CSS3 is used for animations and layouts; scalable vector graphics (SVG) for whiteboard and annotations; localStorage for user configuration; iframe for modular widgets; desktop notification for incoming or missed call; and browser extension for presence and click-to-call.

Vclick demonstrates high quality video conferencing with only a browser user agent. The signaling primitives of current VoIP are re-invented, albeit in a simpler form with endpoint driven app logic. The cloud resource server runs on Amazon EC2. Our on premise system has been tried by over 80 users, and cloud one by about 50 users. Our users have gotten started on the system from signup to first successful call in a matter of few minutes. Installing the entire software on a Linux, OS X or Windows machine with Python takes less than five minutes.

The resource server deals with only signaling, not media, similar to a SIP proxy. To compare with SIP, we will define metrics: average message size, message count and types per call, the request capacity per type per second, and simultaneous persistent connections per server. With horizontal scaling and data partitioning, single server performance is secondary.

WebRTC is still in the early stages with several open concerns: whether it will be available consistently across browsers and platforms, e.g., Internet Explorer or iOS. ORTC may be used without changing the widget interface. Although, such factors will affect WebRTC adoption in the future, our effort proves that a complete flexible collaboration system is possible in a pure-web and browser environment using just a light weight resource server and WebRTC-enabled browsers.

References

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

[2] WebRTC 1.0: Real-Time Communication Between Browsers, W3C Working Draft, Feb 2015, http://www.w3.org/TR/webrtc/

[3] J. Rosenberg et al, "SIP: Session Initiation Protocol," IETF RFC 3261, 2002

[4] K. Singh and V. Krishnaswamy, "Building communicating web applications leveraging endpoints and cloud resource service", IEEE International Conference on Cloud Computing, Santa Clara, CA, Jun-Jul 2013

[5] Sites that use or demo WebRTC, WebRTC world, accessed Jul 2015, http://www.webrtcworld.com/webrtc-list.aspx

[6] K. Singh, J. Yoakum and A. Johnston, "Enterprise WebRTC powered by browser extensions", Principles, Systems and Applications of IP Telecommunications (IPTComm), Chicago, IL, Oct 2015