Enterprise WebRTC Powered by Browser Extensions

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

John Yoakum
Avaya Inc.
Cary, NC, USA
yoakum@avaya.com

Alan Johnston
Avaya Inc.
St Louis, MO, USA
abjohnston@avaya.com

ABSTRACT
We use browser extensions to solve two important issues in adopting WebRTC (Web Real-Time Communications) in enterprises: how to integrate WebRTC-centric communication with existing systems such as corporate directories, communication infrastructure and intranet websites, and how to traverse media paths across enterprise firewalls. Vclick is a simple and easy to use web-based video collaboration application that enables click-to-call from other webpages. SecureEdge is a network border traversal system for policy and security enforcement, and consists of a secure media relay that sits at the network border or in the cloud. A browser extension in the enterprise user's device transparently injects this media relay in every WebRTC media path needing to traverse the enterprise network edge to enable authenticated border traversal without help from the websites hosting the WebRTC pages. We attempt to generically support WebRTC in enterprises on a variety of application scenarios instead of creating another fragmented communication island. The challenges faced and techniques used in our proof-of-concepts are likely extensible to other enterprise WebRTC scenarios using the emerging HTML5 technologies.

Categories and Subject Descriptors
H.4.3 [**Information Systems Applications**]: Communication Applications – *computer conferencing, teleconferencing and video conferencing.*

General Terms
Design, Experimentation, Security

Keywords
WebRTC, enterprise communication, secure edge, browser extension, VoIP, video call, firewall traversal, media relay.

1. INTRODUCTION
WebRTC (Web Real-Time Communication) consists of the emerging W3C and IETF standards and the ongoing efforts by browser vendors to enable plug-in free browser-to-browser multimedia communications [1][5]. Enterprise adoption of WebRTC faces several challenges as follows:

Competes with existing communication systems: Enterprise users who are already familiar with existing voice-over-IP (VoIP) phones, conference bridges or other web conference systems, are reluctant to switch to a new technology.

IPTComm'15, October 6-8, 2015, Chicago, IL, USA.

Traversal through enterprise firewalls: Although an enterprise SBC (session border controller) intercepts VoIP signaling channel to detect and open media ports in the firewall, it does not transparently work for peer-to-peer WebRTC media flows where it cannot intercept the secure signaling channels (carried over HTTPS) or end-to-end encrypted media paths [7].

Policy enforcement per user's enterprise identity: Although a website can identify its user, e.g., account name on Facebook or Google Plus, the enterprise IT (Information Technology) department would like to apply policies based on the user's enterprise identity which is different from her website identity.

A few illustrative examples of enterprise policies include: limit bandwidth or duration of a WebRTC flow, disallow information leakage over WebRTC data channel, disallow internal IP address information leakage to external users, record all media flows to and from the enterprise network, or disallow a video stream at certain hours. Although, a website can do these by modifying the signaling data generated by the browser, enterprises would like to enforce these irrespective of which website is used by the user.

We solve these problems by extending the enterprise user's web browser to make it easy to communicate internally and to enforce policies for communications on both intranet and external websites. A browser extension provides an easy way to extend a web browser to intercept web pages and to potentially change them using JavaScript. The benefit is that it can be done without help from the website and applies to any WebRTC flow originating or terminating at the user's browser. Such extensions may be deployed as part of IT software management process or by individual user. Such carefully crafted browser extensions allow enterprise customization of browser capabilities.

We present two proof-of-concepts built using browser extensions: the first one named *Vclick* is an easy to use web-based video collaboration application that enables click-to-call from web pages such as intranet corporate directories, and the second one called *SecureEdge* consists of a media relay that together with a client-side browser extension enables IT policy and security enforcement on the WebRTC media path. These proof-of-concepts are purely browser based using HTML5 technologies [3] independent of legacy VoIP protocols [8].

We present background on WebRTC and related work on its enterprise adoption in Section 2. Section 3 and 4 describe how browser extensions help our Vclick and SecureEdge implementations, respectively. Section 5 lists example use cases in which a browser extension can benefit WebRTC applications. Section 6 has our conclusions and future work.

2. BACKGROUND AND RELATED WORK

2.1 WebRTC notification system
WebRTC enables a web page to establish a peer connection between two browsers or other entities, and transport captured

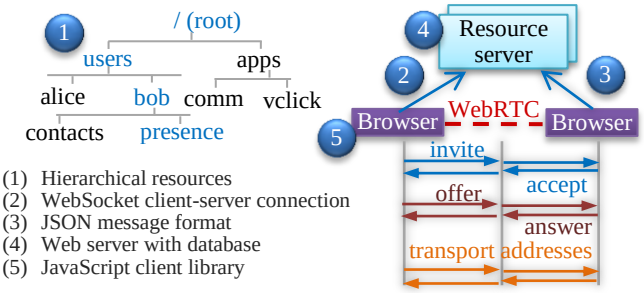

Figure 1. Resource service as a generic lightweight data access and notification system for WebRTC.

media from one to other. It needs a *notification system* to exchange certain signaling data between the browsers involved. A typical audio or video call needs signaling data for a) call control, e.g., invite or answer events, b) session description including list of media types, codecs and their parameters, and c) transport addresses for creating a peer-to-peer media path. Unlike SIP (Session Initiation Protocol) [8] which sends these three pieces of data in a single request-response, a WebRTC application can send them in any order.

We use the *resource service* [2][9] as a data access and notification system for communicating web applications. This is one of the many ways to implement the notification system. Further details are in [9], and are summarized in the five concepts shown in Fig.1. The system allows storing and exchanging data and events on target resource paths, e.g., Alice's browser subscribes to the presence resource of Bob, /users/bob/presence, and gets notified whenever the resource is changed by another client, say Bob's browser.

2.2 Enterprise firewall and WebRTC

Enterprise firewalls typically allow web traffic but block any unsolicited peer-to-peer traffic including that of the WebRTC media path [7], e.g., you cannot always talk to the other person or there is one way media. Fortunately, WebRTC allows an intermediate media relay to solve the problem in many cases. Fig.2 shows a media relay in DMZ (de-militarized zone) that relays UDP packets across the enterprise boundary in a three party call with two internal and one external browsers, all talking on a public website. The pair-wise session negotiation picks the best media path for each peer connection, e.g., direct on the intranet and via the relay across the border.

There are two challenges in this architecture: (1) the web server does not know about the enterprise media relay or its IP address, but the web page in the browser should know about it to use it, and (2) the identity of the user on the public website is often different from her enterprise identity. The second point means that the IT cannot easily enforce per-user policies on the media path at the border. WebRTC aggravates the problem because the signaling data for these media flows are often controlled by third-

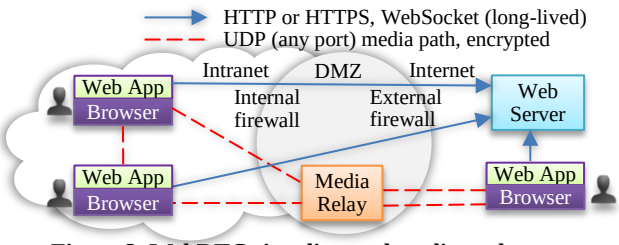

Figure 2. WebRTC signaling and media path across enterprise firewall in a three party call via a relay.

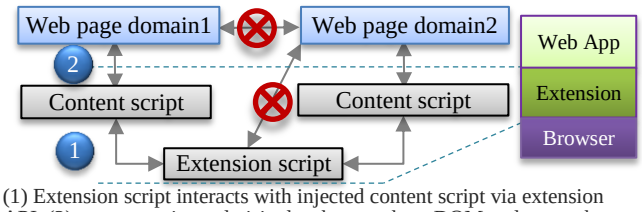

Figure 3. Interaction in Google Chrome.

party websites, and are unavailable to the intermediate firewalls due to the encrypted transports.

2.3 Browser extension and communication

Unlike a plugin that renders some content on a page (e.g., QuickTime), a browser extension extends the functions of the browser and can apply to any web page, e.g., to track visited pages and interact with others on that page. An extension is often written in JavaScript and has limited sandboxed access to the underlying system. It can modify visited web pages, or intercept and change native JavaScript APIs of the browser.

Extensions are browser dependent (different for Google Chrome and Firefox), and often unavailable on mobile devices. Google Chrome has different types of extensions [11]; browser action is a common type that appears as a clickable icon next to the address bar. Browser's cross origin policy prevents direct interaction between the extension script and the visited page, but requires a non-trivial interaction between the extension script and the injected content script using the browser specific APIs, and between the content script and the visited page using events and shared DOM (document object model) (Fig.3). We use such interactions in Vclick and SecureEdge.

Using a Vclick-style browser extension for WebRTC-based video communication is not new [10]. However, our work goes beyond just a video call because the underlying reusable widgets easily integrate with a wide range of enterprise applications including corporate directory, team spaces, video presence, and VoIP. To the best of our knowledge, use of a browser extension such as our SecureEdge extension in applying enterprise policies to and media recording of WebRTC media flows has not been done before.

3. PRESENCE AND CLICK-TO-CALL

Vclick is a web based enterprise collaboration system with pluggable widgets [9] for audio and video calls, conferencing, text chat, file and screen sharing, shared whiteboard, etc. It uses email address as the user's identity. We have deployed it on our intranet as well as in the Amazon cloud.

3.1 Role of a browser extension

The application is divided into two parts: the browser extension and the conversation page. Most widgets such as video call or text

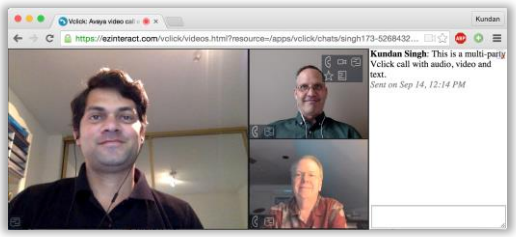

Figure 4. Conversation page in a three-party call with voice, video and text chat. The extension icon next to the address bar reflects the user's presence or call state in that browser tab.

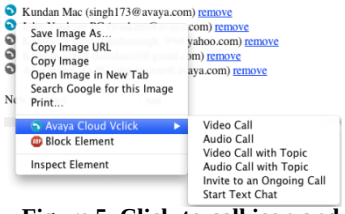

Figure 5. Click-to-call icon and right-click context menu injected by the extension on a visited page.

chat run from the conversation page in separate browser tabs or iframes (Fig.4). The browser extension is required to serve two functions: (1) set user's presence and exchange call initiation events, and (2) modify visited pages to add click-to-call. The browser extension runs in background as long as the browser is running to implement these functions.

The extension runs on Google Chrome on Windows, Mac OS X and Chromebook. It shows an icon next to the address bar, and when clicked allows initiating an outbound video call. An incoming call is notified using desktop notification, and can be answered or declined, or timed out to indicate a missed call. When a call is initiated or answered, the extension opens a conversation page, and keeps track of all active conversations.

Separating call initiation from conversation is not typically found in existing SIP systems, but it makes our software flexible and extensible. If a user lands on the conversation page with the right URL parameters, she joins the conference, irrespective of if the URL was received via the extension's call logic or sent out-of-band on email or other channels. The call initiation does not care which media types are used in a conversation: audio, video, text, whiteboard, notepad, etc. The conversation page takes care of further call control such as participant join or leave, and enables drag-drop behavior, e.g., for call transfer or sidebar conversation.

3.2 Integration with existing web pages

The extension tracks visited pages to selectively modify them, e.g., on our corporate directory pages, it detects all mailto: user@avaya.com URLs and appends click-to-call <img> tags next to them. When clicked, it initiates a call to the target user. The image changes to reflect the target user's presence state.

The browser extension also adds a right-click context menu on the click-to-call buttons, which allows selecting how to reach the target (Fig.5). The extension can be configured to work on any websites beyond just the directory pages, by detecting user identifiers and making them click-to-call'able.

Ability to initiate web collaboration in whatever a user is doing, from whatever device he has, and to interoperate (instead of compete) with existing enterprise communication systems are important in enterprise adoption of WebRTC.

3.3 Interoperation with telephony gear

Besides adding click-to-call buttons, the extension also detects phone numbers to make them click-to-call'able via our WebRTC-to-SIP gateway. For internally hosted Vclick, the media path remains internal. For cloud hosted deployment, we use a hosted STUN and TURN server [5] to allow cross border media flows.

4. ENFORCING ENTERPRISE POLICIES

We present the design and implementation of *SecureEdge*, a border transversal system that applies enterprise policies to WebRTC flows irrespective of – and without help from – the website or web application the user is currently using. The system consists of (1) a secure media relay through which all UDP traffic must flow; and (2) a browser extension in the browsers of the intranet users which intercepts WebRTC to inject the media relay in all peer connections. The first point is particularly important, because without such restriction a user may bring-her-own-device (BYOD) or may use another browser to bypass the policies regarding WebRTC flows across the enterprise network edge.

4.1 Role of the media relay

The media relay could be in the DMZ or in the cloud. The enterprise firewall rules must block cross border WebRTC except with the specific media relay IP addresses. Since encrypted WebRTC flow is not distinguishable, any UDP traffic above port 1024 is blocked. Both the media relay and the client browser extension work together in enforcing the enterprise policies, and hence all UDP traffic not going through the media relay is blocked to prevent bypassing it to circumvent enterprise policies.

The media relay serves two purposes: (a) enable secure and authenticated media flows, and (b) act as a man-in-the-middle of the media flow if needed, e.g., for recording.

4.2 Role of the browser extension

The extension intercepts WebRTC API calls, and delivers the necessary identity and transport information to the media relay. In particular, it does the following tasks transparent to any page that uses the WebRTC APIs in the user's browser:

1. It changes the definitions of the WebRTC APIs available to the web pages running in the browser.
2. It inserts user's enterprise identity in the signaling data.
3. It injects the media relay's IP address in the peer connection, in place of or in addition to any other servers used by the website.
4. It detects any active peer connection, and potentially allows the user to monitor and control it.

The extension exposes a proxy object for the WebRTC peer connection which hides the real object, RTCPeerConnection, of the browser, contains an instance of the real object, and allows modifying certain session and transport data. It also intercepts the WebRTC getUserMedia function that is used to get a local media stream from camera or microphone devices. The web page running in the browser downloaded from a third-party website invokes the WebRTC APIs without knowing that the definitions of those API classes and functions have been replaced. Behind the scenes these API calls go through our proxy objects or functions.

Fig.6 shows how the extension of the internal user's browser interacts with a secure media relay at the edge. Even if only one side of the peer connection has the extension, it can inject the media relay in the media path. Fig.7 shows how the extension intercepts WebRTC APIs on any web page and replaces them with custom processing. Both newer Promise-based and legacy callback-based APIs [1] are supported by the extension.

We use a modified open source TURN relay server [4]. The software maintains per-user long term credentials based on enterprise identities, and applies policies such as maximum bandwidth and call logging. These policies are also applied by the client extension using the intercepted WebRTC APIs.

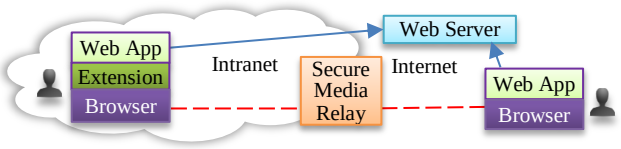

Figure 6. Extension in the intranet browser can inject the media relay on the peer connection media path.

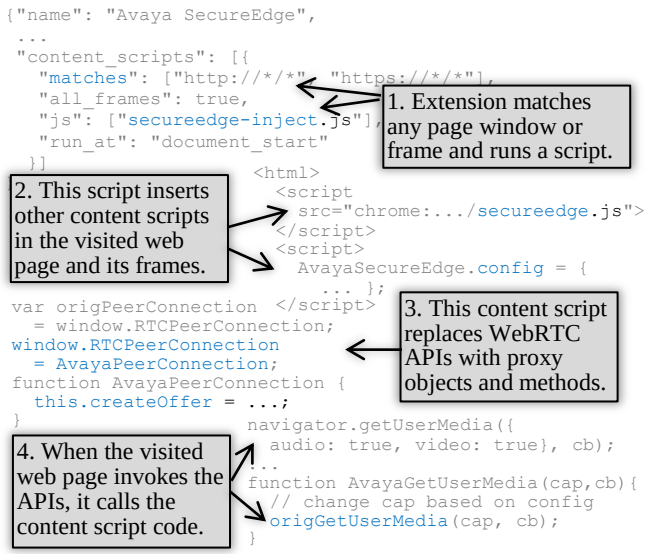

Figure 7. Intercept WebRTC on a webpage by an extension.

4.3 Customization via configuration

Fig. 8 shows the configuration page to set identity and media relay data. This can be preset by enterprise IT on users' browsers. The configuration is stored in the browser's local storage associated with this extension, and is not visible to other web pages or extensions, and not even to the web server.

Identity: We use email address along with per user certificate signed by the organization. Its private key is stored in the local storage, and used for signing requests by this secure identity.

Settings: This is used to filter devices or transport addresses, e.g., if camera is disabled, then the getUserMedia function is intercepted and the "video" capability is removed, or when one or more relayed addresses are specified, then those addresses are inserted when constructing an RTCPeerConnection. A value of "any" keeps the existing list of servers of that type supplied by the web page. The local interfaces attribute is useful in filtering local IP address by intercepting onicecandidate, e.g., "10.0.0.0/8" indicates that only this subnet address is used and other local ICE candidates of type "host" are dropped. This can prevent leaking IP address information such as of VPN.

Figure 8. SecureEdge configuration page.

Connections: This part shows a list of active peer connections on any web page or frame. It also shows the peer identity if available to which the connection is established, and allows controlling it to some extent, e.g., to abruptly disconnect it.

4.4 Applying secure identity

The extension fills the gap between the specification and implementation of secure identity in today's WebRTC [1]. A web page specifies the identity provider domain, and tells the browser to contact it for signing all the signaling data, so that the other end can verify the identity. However, in an enterprise setting, we would like to specify the user's enterprise identity irrespective of what the web page wants.

The extension can insert signed identity in the signaling data. First, the user specifies her identity on the configuration page and creates a secure credential (certificate) issued by the organization. Then the extension uses the associated private key to securely sign identity inserted in the offer or answer or any other signaling data created on any webpage by any peer connection. At the other end, if an identity exists, the extension automatically verifies that it is valid with trusted issuer.

4.5 Informing webpage origin to media relay

Existing web browsers restrict cross-origin resource sharing, e.g., when a web page on origin https://firstserver.com initiates an Ajax request to https://secondserver.com, the request is denied unless the second server explicitly approved the first server. The browser sends an Origin header containing the first origin in HTTP request to the second server.

In WebRTC, similar idea can be used to inform the STUN and TURN servers about the origin of the webpage requesting the service via a new STUN attribute [6]. This is particularly useful when the media relay and the website are operated by different unrelated entities, e.g., a media relay on enterprise edge can apply policies whether to allow or deny the relay service based on which website is initiating the conversation.

In the absence of browser support of STUN Origin, a browser extension can intercept WebRTC API and inform the network edge either out-of-band or via text encoding in STUN username attribute about the website origin that is attempting to use the particular STUN or TURN server.

4.6 Call logging and accounting

Although WebRTC does not have a notion of a call session, we may use heuristics to co-relate the media streams and peer connections on the same web page or website to belong to the same call or conference. This brings a session context to media streams or flows, e.g., for logging and accounting, which is a very crucial policy requirement today with respect to VoIP. Intercepting WebRTC APIs allows us to know when a peer connection is established or terminated, or a page is closed.

4.7 Call recording and server side media processing

Server side recording requires injecting a media server as a man-in-the-middle of the peer-to-peer media flow of WebRTC. The extension intercepts and modifies the necessary signaling data without help from the website, which allows recording WebRTC conversation on any website as described below.

In WebRTC, the peer connection's fingerprint (i.e., hash of DTLS' public key) is exchanged and locked in the signaling data so that each side can verify that the other end is who it claims to be. Intercepting this end-to-end encrypted media flow is not possible unless one has the DTLS private keys. If a media server is inserted in the middle to terminate and initiate DTLS flows, it must also change the fingerprints in the signaling data. Unfortunately, if HTTPS is used to transport the signaling data, a

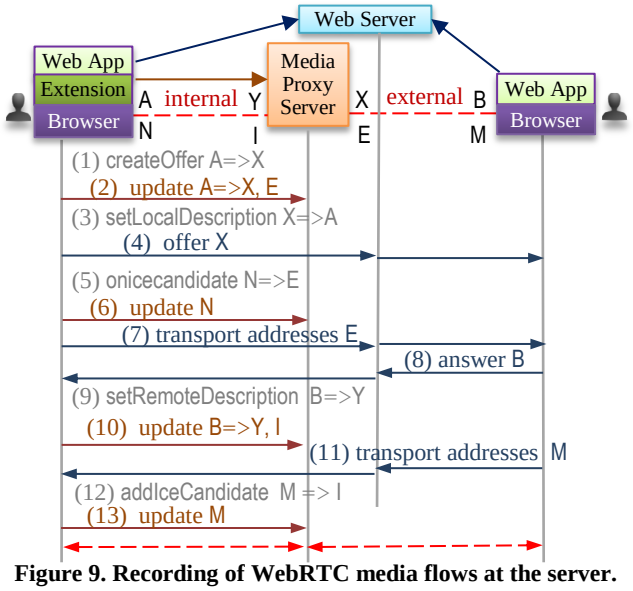

Figure 9. Recording of WebRTC media flows at the server.

web proxy cannot easily change it. Thus, it can only be changed either by the website, or at the end-point in the browser or browser extension, but not by an intermediary such as an SBC.

The extension intercepts WebRTC APIs createOffer, createAnswer, setLocalDescription, setRemoteDescription, onicecandidate and addIceCandidate (and in future any other relevant functions) to substitute the fingerprint and transport addresses. This tells the webpages that the two browsers are talking to each other, but tells the two browsers to talk to the intermediate media server. In Fig.9, if the two browsers generate their local sessions as A and B containing their DTLS fingerprints, the extension changes them so that the two web pages know there sessions as X and B, respectively, and the first browser uses local session A and remote Y, whereas the second browser uses remote session X and local B.

The detailed message flow is described below from the first browser's perspective that uses the extension to inject the media server in the path of an outbound peer connection. A similar flow for the inbound peer connection by intercepting createAnswer instead of createOffer is trivial to derive from below.

1. When the webpage invokes createOffer, the extension intercepts it, gets the local session description from the browser, say A, but delays returning it to the webpage.
2. The extension sends A to the media server, via an out-of-band API, to replace the DTLS fingerprint with that of the external side DTLS port of the media server. It gets the updated session, X, and external side address, E. It returns X to the webpage in the response callback of createOffer.
3. The webpage calls setLocalDescription with X. The extension changes it back to A, so the browser sees its own fingerprint.
4. The webpage sends the offer X to the other endpoint using the signaling channel.
5. The extension intercepts onicecandidate callbacks on the peer connection. It replaces the transport addresses, N, with that of the external side DTLS address, E, of the media server, and suppresses any more unused callbacks.
6. The extension sends N to the media server, so that the server can reach this endpoint on media path.
7. Eventually, the webpage sends E to the other endpoint via the signaling channel. Thus, the other end sees only the media server's external address.
8. The other end creates its answer session, B, via createAnswer, and sends it to this webpage on the signaling channel.
9. When this webpage receives the answer B, it invokes setRemoteDescription. The extension intercepts it and delays telling the browser.
10. The extension sends B to the media server to replace the DTLS fingerprint with that of its internal side DTLS port. It gets the updated session, Y, and internal side address, I, of the media server. It then invokes setRemoteDescription with Y to inform the browser about the remote session as Y.
11. Any received transport addresses, M, from the other endpoint are passed by the webpage to the peer connection in addIceCandidate (or similar function).
12. The extension intercepts this and replaces M with the internal side address, I, of the media server. If the firewall blocks all UDP transport except to/from the media server, most of the remote candidates become useless, and only the path via the media server works.
13. The extension sends M to the media server, so that the server can reach the other endpoint on the media path.

At the end of these steps, this browser's peer connection has local session A and remote Y, and the other browser has local B and remote X. Note that X and Y contain the DTLS fingerprints of the media server ports, on external and internal sides, respectively. Thus the two browsers can establish DTLS media path with the media server without the application's knowledge.

Instead of using host transport addresses representing the media server, one could use TURN relay candidates, where the TURN server is co-located with the media server. In that case RTCPeerConnection is intercepted to inject that TURN server and remove any other application provided STUN/TURN addresses. Similarly, onicecandidate and addIceCandidate are intercepted to suppress any other addresses other than this TURN server.

This injected media server in the DTLS flow can decrypt and do media processing including transcoding, media recording, or inserting voice prompts or video advertisements in the flow.

In another approach, the extension can record the local camera and microphone devices every time getUserMedia is invoked by creating a forked media path to a media server. However, this is not as robust or as effective as man-in-the-middle recording at the media server, because the user may not be in a call every time local camera or microphone is captured, or she may be able to suppress sending the recorded file to the IT server.

In future, we expect that the client-side media stream recording and/or appropriate server-side recording hooks in the emerging WebRTC APIs will avoid the need for such complex manipulation of the media path. In the meanwhile, a browser extension can accomplish the desired behavior for enterprises that want to record all media flowing across their network edges.

4.8 Limit per-user bandwidth

Once WebRTC allows limiting media stream bandwidth, the extension can apply such policies by modifying the bandwidth attribute in the WebRTC signaling data, and the media relay can enforce the upper limit, dropping packets when needed.

5. WHEN IS AN EXTENSION NEEDED?

We have shown how a browser extension can transparently enforce enterprise policies or insert communication widgets on any third-party websites. The key is "any third-party websites",

e.g., an extension is not needed if the website owner wants to enforce policy like call recording on a contact center website.

An extension can also facilitate screen and app sharing, preserve peer connection on page reloads, or enforce specific media or device constraints as illustrated below:

1. Due to security concerns, WebRTC-enabled browsers such as Google chrome do not allow screen or app sharing on any webpage, but require that the request be made from a native app or a browser extension.
2. Keeping the peer connection state in the extension can preserve it on page reload or navigation within the same website avoiding a media path reconnection.
3. It can inject attributes in getUserMedia to take advantage of user's high definition (HD) camera or can alter the list of codecs negotiated in the signaling data, e.g., to prefer G.711 over Opus voice codec.
4. It can display all received video streams and allow drag-and-drop to external devices or browser tabs, e.g., to display a video in full size, irrespective of the size constraints imposed by the web page.
5. It can intercept and disable all WebRTC data channel APIs, to avoid leaking proprietary information or remove sensitive IP address or other information from signaling data.
6. It can replace or create APIs on browsers that do not support WebRTC or have non-standard APIs, e.g., by using plugins and/or by using server side functions when needed.

An extension's ability is restricted to the JavaScript APIs exposed by the browser. Currently, raw or encrypted video data is not available in JavaScript; hence an extension *cannot* perform speech recognition or video frame alteration in a received media stream, or use text-to-speech to generate outbound media flows.

6. CONCLUSIONS AND FUTURE WORK

We have shown how to use browser extensions to solve problems in enterprise adoption of WebRTC. Using a network-only element does not work for applying enterprise policies to WebRTC traffic. Hence, either the web page or the browser should work in co-operation with the IT policies. Modifying the browser using an extension allows us to transparently intercept WebRTC APIs and apply policies. In the future, browser vendors may include this feature natively in their browsers, e.g., to let the end user or enterprise policies inject network elements in the media path of any WebRTC traffic.

Seamless integration with existing systems will likely promote WebRTC's adoption. The Vclick browser extension enables communication from other pages such as corporate directories without help from those websites. It allows interoperability with existing telephony gear. Integration with some other enterprise systems becomes trivial and has been experimented with, e.g., send the call invite via email or instant message with a clickable link to the conversation page. We have also added support for screen and app sharing, shared white-board, and notepad with real-time text updates. We have been using on-premise Vclick internally in a small group since early 2013, and the cloud hosted version since early 2014. The underlying reusable widgets of Vclick have influenced our other research projects on WebRTC for desktop as well as mobile. We continue to use the Vclick widgets and components in our emerging research projects.

A browser-to-browser call has minimum load on the server due to peer-to-peer media path. Although, we have implemented automatic failover and scalability measures for the signaling channel, those topics are out-of-scope for this paper as we focus here on novel ways to use browser extensions. Performance evaluation of the Vclick extension on quality of service and formal verification of the SecureEdge message flows from a security perspective are for further study.

In the absence of browser extension support on mobile devices, we are exploring alternatives. Our Android version of Vclick uses HTML5 and Apache Cordova. With the help of Android web-intents, we can launch it to dial out a target user from click-to-call on web pages similar to the browser extension's click-to-call behavior on desktops. However, transparently intercepting WebRTC APIs on a mobile browser is challenging, and may be solved using a custom browser or by use of a web proxy.

We have described our implementations and listed various challenges. Our future work targets new ways to modify and utilize WebRTC in enterprises, including on mobile devices, and keeping up with the progress in standardization and browser implementations.

7. REFERENCES

[1] Bergkvist, A., Burnett, D.C., Jennings, C. and Narayanan, A. 2015. *WebRTC 1.0: Real-Time Communication between Browsers*. Working Draft. W3C. Feb 2015. http://www.w3.org/TR/webrtc/

[2] Davids, C. et al. 2011. SIP APIs for voice and video communications on the web. In *Proceedings of the 5th International conference on principles, systems and applications of IP telecommunications* (IPTcomm'11, Chicago, IL, Aug 2011). DOI:10.1145/2124436.2124439.

[3] Hickson, I. et al. 2014. *HTML5: a vocabulary and associated APIs for HTML*. Recommendation. W3C. Oct 2014, http://www.w3.org/TR/html5/

[4] High performance free open source TURN and STUN server implementation. Accessed Jul 2015. https://github.com/coturn/rfc5766-turn-server/

[5] Johnston, A.B. and Burnett, D.C. 2014. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*, third edition, Digital Codex, ISBN 978-0985978860.

[6] Johnston, A., Uberti, J., Yoakum, J. and Singh, K. 2015. *An origin attribute for the STUN protocol*. Internet draft. "Work in progress". IETF. Feb 2015.

[7] Johnston, A., Yoakum, J. and Singh, K. 2013. Taking on WebRTC in an enterprise. *IEEE Communications Magazine*, Vol.51, No.4, Apr 2013, doi:10.1109/MCOM.2013.6495760.

[8] Rosenberg, J. et al. 2002. *SIP: Session Initiation Protocol*. RFC 3261. IETF. Jun 2002.

[9] Singh, K. and Krishnaswamy, V. 2013. Building communicating web applications leveraging endpoints and cloud resource service, In *Proceedings of the IEEE 6th International Conference on Cloud Computing* (CLOUD'13, Santa Clara, CA, Jun-Jul 2013). IEEE Computer Society. pp. 486-493. DOI:10.1109/CLOUD.2013.39.

[10] Twelephone: putting video calling in Twitter feed. http://blog.twelephone.com. Website. Accessed Jul 2015.

[11] What are extensions? Accessed Jul 2015. http://developer.chrome.com/extensions. Website