HED: Web Based Solution for Effective Real Time Communication

JinoWiseson, A.¹, Suthan. A.²
M.Tech IInd year Information Technology Dr.SivanthiAditanar College of Engineering, Tiruchendur¹
Assistant Professor Information Technology Dr.SivanthiAditanar College of Engineering, Tiruchendur²

Abstract:
Real-time interactive communications have become critical for business, e-commerce, e-government and education. Teleconferencing systems have been usually deployed using proprietary and expensive systems. Overlay networks are often used to support multi-point teleconferencing and provide services unavailable in the underlying physical networks. With the advent of WebRTC, real-time communication applications can be implemented directly in the browser without installing custom applications or additional plugins. It is possible to use the HTTP / Web Socket and RTC API via a Web server, in order to perform real-time video communication in all Web browsers. In all environments that are connected to the network, it is available at all terminals that you can use a Web browser and real-time video communication. Web Real-Time Communication (Web RTC) is an upcoming standard that aims to enable real-time communication among Web browsers in a peer-to-peer fashion. In this paper we are aiming to present the detailed review over WebRTC framework. We are presenting the standard and technology used of WebRTC; different methods used in WebRTC, bandwidth allocation scheme discussed which is used for video collaboration under real time environment. This project will support two peer of different with help of heterogeneous. It will work with help of the peer to peer architecture and Web Socket that will help to avoid video buffering.

Keywords: Browser, Heterogeneous, HTML5, Peer-to-Peer, Real-time communication, WebRTC, Web Socket.

I. INTRODUCTION

Google started early the effort called WebRTC in order to build the real time video collaboration media engine for all the available internet web browsers. Since 2002 Global IP Solutions (GIPS - formerly Global IP Sound)) wrote object-code for the likes of Nortel (Avaya), Webex (Cisco), Yahoo and IBM to support their PC-based telephony applications. GIPS Google bought in 2010 for $85m. In 2011, the Global IP Solutions (formerly Global IP Sound) acquisition using the acquired technology, an open source version of Google WebRTC media engine is built and implemented into Chrome. In a browser with WebRTC , a Web services application WebRTC Now another device or using RTP for WebRTC media server to create a real-time voice or video connection can instruct the browser. Signalling and protocol standards from the W3C and IETF API for application developers is coming from, so communication can be defined and not just SIP and VoIP systems developers with a small number of sellers, by millions of developers have Java Script. [5] First WebRTC enabled browser, Chrome and Mozilla, in fact, hidden behind a flag WebRTC to Chrome current browser, you will come out later this year, but the ability to test and test [1] [2] [3] [4]. Common Web browsers, with an interactive component common RTC API to a website using an application must be able to be added. Web chat button is over video on the RTC websites. Hackathon events early efforts to great effect combined with Web RTC GL: live feeds crossword puzzles, playing with musicians throughout the Web, and a host of ready-made effects being made in [5]. Web applications without mediation with their peers will allow sharing data as even more important; on the horizon could be peer-to-peer data channel [6]. In this paper we are discussing WebRTC framework, literature survey over the same, also discussing the technology and standard of WebRTC, bandwidth allocation scheme is discussed. In section III we are presenting the detailed technology for WebRTC.

With multimedia becoming the norm for exchanging information in today’s world, the security of commercial multimedia applications has assumed critical importance. For instance, enterprises with distributed locations having their business meetings via video conferencing, is now a commonplace. Having an intruder intercept the path of data transmission and thereby gain access to the information being transferred can lead to horrendous situations especially in scenarios wherein sensitive data is being transferred. Another related domain is the video-on-demand application wherein certain privileged users are granted access to receive the benefits of the service. To ascertain that the signal is not intercepted on its transmission path and hence prevent the misuse of the service, encryption can be used. Such applications need stringent encryption algorithms for which the incurred expenses for cracking the encryption in terms of the cost should be more than the legal access to the service. This is to deter the misuse of the service. In addition, the time needed to crack the code should be significant to ensure a sufficient level of security. Processing of large video files involves a huge volume of data. The codec, storage systems and network need high levels of resource utilization, i.e., processor time. Complex algorithms for encryption will only aggravate the problem and increase latency. Thus, the algorithm needed needs to be both secure as well as fast. In this paper, we propose a system using the RSA algorithm. The RSA algorithm involves three steps: key generation, encryption and decryption.

II. BACKGROUND AND RELATED WORK

Before describing the interoperability between SIP and WebRTC, we give a brief background on these systems and their differences from the interoperability point of view.

A. What is SIP?
SIP [1] is the IETF standard for establishing, managing and terminating Internet sessions including voice and video calls
and conferences. As shown in Fig.1 (a), a SIP system uses other standards such as SDP (Session Description Protocol) for offer/answer of session negotiation, RTP (Real-time Transport Protocol) for media path transport, RTCP (Real-time Transport Control Protocol) for feedback and control of media path, optionally SRTP (secure RTP) with keys negotiated in SDP for media path security, and optionally ICE (Interactive Connectivity Establishment) for traversal through intermediate NATs and firewalls. The dotted red-line separates what is programmed by the application developer and what is provided by the platform.

B. What is WebRTC?

WebRTC represents the family of emerging standards within the WebRTC working group in W3C [3] and the IETF RTCWg working group [2] to enable end-to-end browser communication for real-time media. Please refer to [4] for an overview of WebRTC. It reuses existing standards such as mandatory SRTP (Secure RTP) for media transport, and mandatory ICE (Interactive Connectivity Establishment) for traversal through NATs and firewalls. The signaling messages are browser independent and are left to the application developer who would typically use HTTP (Hyper-Text Transfer Protocol) and WebSocket (WS) [5] for exchanging call control and session description information among the participants. The WebRTC API proposal [3] uses SDP, which enables an application developer to use it as is, or transform it to webfriendly JSON (JavaScript Object Notation) or XML (eXtensible Markup Language). Optionally, the application can include a SIP implementation in JavaScript and reuse all the features provides by SIP.

C. What is WebSocket?

WebSocket (WS) is an IETF protocol and an HTML5 API that allows creating a bi-directional client-server connection from the JavaScript code in the browser to the web server. To traverse web proxies, it uses HTTP to initiate the first request and subsequently upgrades to a persistent connection via additional handshakes. Once the connection is established, it allows sending any data with packet boundary in either direction over TCP.

D. Related Work in Interoperability

In the early days of WebRTC, the IETF rejected having SIP in the browser and left the signaling to the application in JavaScript. Subsequent attempts to interwork between SIP systems and WebRTC enabled browsers fall in two categories: translation at the gateway or implementing SIP in JavaScript. SIP already supports several underlying transports such as TCP and UDP, and can be extended to support WebSocket as yet another transport, if needed [6][7]. This is now available in popular SIP proxy servers such as Kamailio and Office SIP. It lets developers implement SIP in JavaScript while promoting end-to-end media path if possible [8][9][10]. Translation at the gateway requires that both the signaling and media path go through the gateway [11][12]. This approach is being adopted by service and application providers to enable yet another way to connect to the service infrastructure. We compare these two approaches in the next section.

III. ARCHITECTURE AND PROCESS OF THE INTERCOMMUNICATION SYSTEM

The WebRTC/SIP intercommunication system consists of browsers, WebRTC clients, the WebRTC server, the WebRTC gateway, the IMS network environment and SIP terminals. WebRTC clients are running in the web browser environment, which consist of the business logic module and the interface UI module. The business logic module is responsible for management and control of media sessions, which consists of JSEP protocol implemented in JavaScript and the signaling protocol stack (ROAP, SIP, and XMPP). The interface UI module is responsible for interacting with users. WebRTC clients use universal web technologies such as HTML, CSS and JavaScript, and have good compatibility and portability. The WebRTC server is responsible for management of WebRTC sessions in the signaling plane, which has the functions of routing signaling messages, maintaining session state machine, and handling session exceptions. When a message is received, the WebRTC server will route the message to a local WebRTC client connecting with the WebRTC server or a SIP terminal in IMS network according to the identifier of the callee. The WebRTC gateway includes the signaling module and the media module. The signaling module is responsible for the conversion of signaling protocol between WebRTC clients and SIP terminals, and the routing of signaling message. The media module is responsible for establishing P2P communications with WebRTC clients and SIP terminals, transforming between SRTP and RTP, decoding and encoding media stream according to the codecs of both sides, and sending the processed media stream from one side to the other side. The media gateways establish socket connections with the signaling gateway, and the signaling gateway maintains sockets of all media gateways. When a new session request arrives, the signaling gateway evenly distributes sessions to different media gateway, so that load balancing of themedia gateway for media session can be achieved. In addition, the deployment of media gateways has high scalability. When the load of current media gateways is too large, we can dynamically increase the number of media gateways by the way of deployment to improve the processing capability of system for media sessions. When a media gateway is down and disconnects with the signaling gateway, the signaling gateway will no longer distribute a new session to the media gateway.

IV. VIDEO ENCRYPTION USING RSA ALGORITHM

RSA is a public-key cryptography algorithm, based on the presumed difficulty of factoring large integers, the factoring problem. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described it in 1977 [25]. The RSA algorithm consists of three steps: key generation, encryption and decryption. A key is a piece of information that determines the functional output of a cryptographic algorithm. Without a key, the algorithm would be useless. In encryption, a key specifies the particular transformation of plaintext into cipher text, or vice versa during decryption. There are two keys in RSA, i.e. Public key and Private key. The public key is known to everyone and is used for encrypting the messages; these messages can be decrypted only using the private key. The public key consists of the modulus n and e (encryption exponent). The private key consists of the modulus n and d (decryption exponent), the decryption exponent has to be kept secret along with p, q and φ(n), using which the decryption exponent can be calculated.

V. 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. The application
is divided into two parts: the browser extension and the conversation page. Most widgets such as video call or text chat run from the conversation page in separate browser tabs or iframes. 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.

**Figure 1. Presence and click-to-call**

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.

**VLENFORCING 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.

**Role of the media relay**- 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.

**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 detects any active peer connection, and potentially allows the user to monitor and control it.
4. 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.

The extension exposes a proxy object for the WebRTC peer connection which hides the real object. RTC Peer Connection, of the browser, contains an instance of the real object, and allows modifying certain session and transport data. It also intercepts the Web RTC get User Media 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. 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. 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 2. Role of the browser extension**

**Call logging and accounting**- Although WebRTC does not have a notion of a call session, we may use heuristics to correlate 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.

**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 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.

Figure 3. Call recording and server

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, whereas the first browser uses local session A and remote Y, whereas the second browser uses remote session X and local B.

VI. PROPOSED SOLUTION

Having used open source components mentioned above, we are able to build a laboratory prototype environment with an interconnection to a real SIP network. Platform is built up using following software components (Fig. 4). As a WebRTC signaling server we are using fast and efficient Kamailio SIP proxy server which is supporting the Web Socket protocol through one of its modules. As a media gateway we are using the webrtc2sip gateway. As a web server we are using the Apache web server. And finally as a WebRTC client we are using own WebRTC SIP application, which was tested in Chrome (ver. 33.0) and Firefox (ver. 28.0) web browsers.

Figure 4. Chrome (ver.33.0) and firefox (ver.28.0)

We realized several interoperability testing scenarios focusing on the different interconnection cases. The exchange of signaling messages has been captured, analyzed, and evaluated to proof the interconnectivity. We identify following issues. The Chrome browser version 33.0 supports SRTP-SDES, and does not support DTLS-RSRT. Firefox supports DTLS-SRTP. Jitsi SIP client support ZRTP-SRTP. Sessions initiated between two peers of the same type (Chrome either Firefox) works fine. Signaling messages flows over WS to the Kamailio server and media flows directly between browser peers. Sessions between different set of peers (Chrome vs. Firefox) is more complicated, as mentioned browsers does not support the same key exchange mechanisms (SRTP-SDES vs. DTLS-SRTP). The implementation requires the presence of a media gateway (webrtc2sip). Once using the media gateway we need routes signaling message flows over the webrtc2sip entity too. Initiating a session between the WebRTC client and a vanilla SIP client (jitsi in our case) requires the presence of a media gateway too, as the Jitsi client does not support SRTP-SDES neither DTLS-RSRT. Reflectors are also connected to the BFCP server, but they do not perform any floor control operation. In fact, reflectors only receive reports about the state of the floors with a twofold purpose: to know the clients that are allowed to generate media streams and to enforce the floor policy by rejecting the streams of the clients not following the floor control policies. Clients should receive information of the BFCP conference during the session establishment in order to contact the BFCP server and discover floors. BFCP conference data can be described using SDP, so the same SDP message can be used to establish media and BFCP sessions. Thus, the SDP offer generated by the RP handshake process contains the BFCP conference description and the WebRTC media configuration.

VIII. 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. 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 Secure Edge 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. There are several things that that could go wrong in the true motivation of SIP in JavaScript. It has a dependency on a consistent and simple WebRTC standard – what if some browser vendors do not implement WebRTC? What if there are browser backdoors that prevent creating cross-browser applications? The choice of audio and video codecs by different browser vendors could force use of an in-network transcoder or media gateway. Finally, if the vendors cannot figure out how to make profit – especially with the risk of opening up the JavaScript source code, and the lack of control over the endpoint application – they may not adopt this.

IX. REFERENCES


