Signaling Mechanism Design for WebRTC

Adem ATALAY
Netaş Telekomünikasyon A.Ş., Pendik, İstanbul, Turkey

Abstract

WebRTC (Web Real-Time Communication) is a new technology that makes the web browsers and mobile applications capable of real time communication even if the peers are connected to their private networks. The connection can transport audio, video or data from one another. To create a connection between peers initial signaling must be negotiated through a signaling mechanism. This article offers a new mechanism design that is different from regular signaling servers to decrease the network overhead and let peers make signaling on demand especially for mobile applications.

Key words: WebRTC, peer-to-peer, real-time, communication

1. Introduction

A WebRTC session is a peer-to-peer session and initiating a connection requires more work than opening other kind of peer-to-peer connections such as WebSocket, REST or XHR [1]. There is an assumption for establishing these connections that the server, which a peer will connect, is located on a publicly available IP address. For a WebRTC connection, peers can be located in their private networks and they have to tell their capabilities and available connection routes to each other. WebRTC lefts this notification process to application layer which means there can be a lot of signaling mechanisms to achieve this.

A well-known mechanism for signaling is using a specialized signaling server. All of the peers send notifications to the server, which they are connected to, and server forwards them to related peer. Notification requests can be done over WebSockets [2] or HTTP protocol such as REST for sending and long polling [3] for receiving. Both of these ways need extra effort or power to receive notification messages. For example, WebSocket needs to be alive and it cannot be used on demand or long polling needs to make a HTTP request a lot of time in a short period which consumes bandwidth.

This paper offers a new mechanism for signaling which uses WebSocket or HTTP like others but these connections will be used on demand. Mainly, the aim of this work is saving the effort for keeping sockets alive and creating a bandwidth-efficient signaling mechanism.

The rest of the paper is arranged as follows: Section 2 mentions commonly used signaling options for WebRTC applications; Section 3 offers the new technic for bandwidth-efficient signaling and discusses the variations of this mechanism; Section 4 presents some constraints of the mechanism. Some works published in last four years is considered in Section 5 and Section 6 contains the conclusion.

*Corresponding author: Address: Mehmet Akif Mh. Elamış Cd. Cahit Sıtkı Sk. No19/A12 Ümraniye, Istanbul 34744 Turkey. E-mail address: adematalay@gmail.com

2. Signalling Options

One of the essential parts of a WebRTC application is signaling. Not only in WebRTC but almost every peer-to- peer applications make signaling which is used for exchanging events with remote peers. There is a common language that all web applications can speak named HTTP. Although, HTTP is one directional, some methods are used to use this protocol for bi-directional communication. As an official Internet standard, WebSocket is recently used for bi-directional message exchange. Commonly used signaling models work with HTTP request or using direct socket to signaling server.

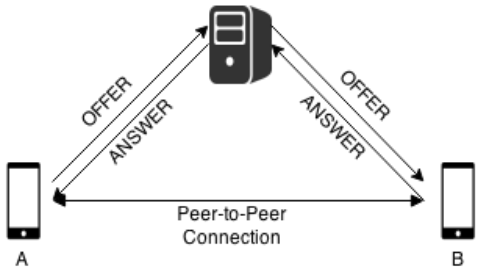

Figure 1. Common use of Peer-to-Peer signaling model

In Figure 1, A and B are connected to signaling server via a socket connection or sequential HTTP requests. A sends his re- quest which has the offer and identity of other peer, signaling server sends this offer to B. Answer is transferred through the same way to A and peer connection is established between A and B.

There is no standard signaling protocol for transferring offers and answers. Some commonly used signaling mechanisms and their issues are as follows:

2.1. Long-polling

Web applications, which are using live content, must get events to refresh their state. Unfortunately, servers without browser request or WebSocket connection cannot update web pages. Long polling which is a Comet [6] like mechanism makes requests to the server in an interval to get events. Therefore, it cannot receive all events at the same time when event happens, servers save this event until next poll. This means there are a lot of empty requests to the server which lead to a big overhead on network if no event rises. Intervals can be set according to the trade-off between efficiency and fidelity. Setting short interval increases the fidelity but decreases the efficiency.

To minimize latency and use of network resources server responds to a request when an event occurred [7]. After client receives a response, he sends new long poll request immediately. Server holds this request open until new information, event or timeout has occurred.

Long polling has some issues that must be considered before using. Header overhead is one of them. Since long polling is a complete HTTP message, it uses full set of HTTP headers [7]. If server has infrequent events, the headers can represent a large amount of data that can cause an unacceptable burden on the server and the network. HTTP is carried over TCP/IP, packet loss and retransmission can occur on non-persistent connections. While for persistent connections long polling works with three network transits that are response, next request and response, packet loss needs more than three networks transits. This increases the maximal latency. The client opens a connection, sends a request, receives the response and finally closes the connection. This cycle runs in infinite loop and opening a connection need some connection establishment time and resource to be allocated. If the server or client is under load, it causes them to run slowly and they queue the messages that significantly reduces the efficiency.

2.2. HTTP Streaming

Streaming mechanism is a kind of long polling that keeps the connection open indefinitely. Even if an event is transmitted to client, connection will not be closed. While this mechanism solves some issues of long polling, it has some own issues. HTTP streaming will not work with some network intermediaries such as some proxies or gateways because intermediaries cache the response before sending to the client. In addition, client buffering and framing techniques can be listed as other issues of streaming [7].

2.3. Sip over WebSocket

In web-based applications usually two-way communication is enabled by WebSockets [4] [2]. This bidirectional and full-duplex connection starts with HTTP/HTTPS protocol and upgrades this protocol to WebSocket during client and server's handshake [8]. Over this connection text and binary data can be transferred in full-duplex mode. Since, minimum data frame size is two bytes, this connection technique does not consume network traffic when it is idle. Therefore, using specially formatted data can do signaling between peers. SIP over WebSocket is like WebSockets, only instead of using special formatted messages; one uses SIP messages in notifications. All clients are connected to server over WebSocket and they make signaling over the servers.

The WebSocket connection must be kept alive which is hard for especially mobile applications for receiving and sending notifications. Server or client keeps the WebSocket connection open by sending ping frames [2]. Loosing connection for a while can lead some notification miss thus systems need ways of detecting failure and resuming sessions to avoid data loss.

3. New Signaling Mechanism

There are mainly two different commonly used mechanisms for getting live notifications in web applications. Long polling makes requests to server in an interval, therefore even if network is down for a while, the notification will be pushed when connection established. On the other hand,

WebSocket connection can miss some notifications according to server implementation but it has less network overhead and latency according to long polling. Since web applications need real time notifications, both of them are not used on demand.

WebSocket and Long-polling mechanism for signaling is used as a client. Beside these connections, peers can have a socket or HTTP server and they can make request directly to each other. For sure, there have to be a common server in this design but this server is not responsible to send the notification to them and peers do not need to care about the connection between them and the server. It is enough for server to tell the IP address and port number to the peer who requests the information of other end. The clients in this system should tell their addresses to the server when addresses are changed.

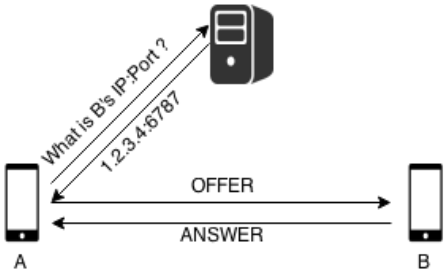

Figure 2. New Peer-to-Peer signaling model

In Figure 2, A and B have their web servers and they inform the server for their reachable IP and port number. The server holds the clients addresses and works like a DNS server. When A wants to establish a peer connection with B, A requests B's address to the server then it sends the first signaling offer to B. B knows the reachable address of A from the request package header and responds its answer to A and peer connection is established. Beside A and B's web servers, they can also use WebSocket server. One peer can open a WebSocket to others' server to send a signal on demand.

Listening a port has no overhead on networks while WebSocket and Long-polling have. A typical http request and its response have approximately 800 bytes overhead and WebSocket has 2 bytes in a message [5]. If 1,000 clients receive one message per second on a WebSocket, network throughput will be 16,000 bits per second that is reasonable for any network. There will be ping-pong messaging even if no message received or sent over WebSocket. If 1,000 clients poll every second on a Long-polling connection, network throughput will be more than 800,000 bytes per second which is more than 6,4 Mbps. On the other hand, in this offered mechanism if no one requests a connection to a web server which are running on every client, there won't be any request therefore no network overhead will be counted.

The new mechanism uses HTTP server running on clients thus it comes with HTTP known issues such as connection establishment time, resource requirement for request and response allocations and security. Secure connections with HTTP protocol are established with usage of SSL certifications [10] [11] and HTTP over TLS [12]. Valid SSL certificates are important for secure

connections. In this new approach it is hard to give a valid SSL certificate to all clients because of the variability of their addresses. They can generate and use their self-signed SSL certificates. However, usage of invalid or self-signed certifications for SSL leaves connections vulnerable to man in the middle attacks [13]. Therefore, if high security is needed, this approach is not recommended to use.

4. Constraints of The Mechanism

Since there is no standard signaling mechanism for WebRTC applications, developers can choose a mechanism which best suits their needs. The offered mechanism can be a new option for developers. Like commonly used signaling mechanism this mechanism has some constraints.

4.1. Secure Connectivity

This mechanism is not applicable for highly secure systems because security is the most important constraints of this for using. The systems that need high security should use well-known methods for HTTP servers like SSL or TLS. The new mechanism is offered for especially mobile devices and they can have variable addresses that can be changed frequently. Therefore, these mobile devices cannot have valid SSL certificates. However, this security requirement can be handled by different ways like using self-signed certificates even if they are less secure.

4.2. Reachability Behind NAT

The mechanism offers an application to have a webserver inside, however webservers should have a public IP address to be reachable. If the application is connected to Internet directly which is not commonly used and have a unique IP address other peers can send requests to it directly. Usually, clients are not connected to Internet directly, they are connected to a private network and this network is connected to Internet over a router. Network address translator (NAT) is a method by which IP addresses are mapped from one realm to another [15]. In a private network a lot of peers can browse on Internet with same public IP address by NAT.

A peer behind NAT does not have a public IP address thus they are not reachable by peers outside the private network. They can use static NAT or port forwarding to be reachable but these methods are also not suitable for mobile peers. To keep the connection alive generally clients send alive messages or frequent request to central server. Network address translators open an address or port for outgoing requests and this tunnel is closed if it is idle for a while. Since networking operations consume too much power, the server can be made responsible for this task if all clients have a webserver inside. The central server can keep this tunnel open by pinging the client even if client does not handle these requests. Time interval for this keep-alive message can be chosen according to dynamic NAT table timeout values [18].

Some kinds of NATs, full-cone NAT, assign a port to the client with first outgoing request. Everyone can reach the client from the public IP of the router and this port number. Therefore, in

this case peers can start signaling with only IP and port information. However, other NATs transfer the request that is coming from only the remote server IP that means peers cannot send requests directly to the clients. Since only the server can reach the client, other peers can start signaling over the server for these kinds of NATs or firewalls.

5. Literature Review

In most of the real-time communication use-cases executed on browsers are web applications for computers. However, there are some use-cases where at least one of the end-user clients is of another type such as mobile devices [21]. In that cases, not only the reliability but also power and network consumption of the system become important to consider.

The general idea behind WebRTC is handling the media path and transportation while leaving the signaling plane to the application layer [19]. In the last years, some approaches are published for hiding complexity of WebRTC's signaling process such as Javascript Session Establishment Protocol (JSEP) [20]. JSEP provides an interface to deal with session descriptions (SDP) [14] that is carried via any signaling mechanism. Offered mechanism in the paper can be one of the suitable ones for mobile devices.

6. Conclusion

The growing new real time communication technology that named WebRTC makes easy to develop and use real time media transportations between individual peers. This communication begins after exchanging session descriptions (SDP) between peers and the mechanism for exchanging SDPs is left to developers. There is no standard signaling protocol for achieving this [9]. The standard mechanism must have some requirements such as two-way communications, secure transmission and authentication support. Even if some protocols have efficiency issues, they are commonly used for this purpose and they are becoming standard like Jingle, Websocket with SDP, Websocket with SIP [9].

In this paper, a new mechanism is offered for making signaling available on demand. By the way, this mechanism is designed for saving the effort for keeping sockets open and creating a bandwidth-efficient signaling mechanism. This is suitable for systems that need medium security and low bandwidth. Battery is crucial for mobile devices and network operations consume too much power, the motivation of this paper is reducing this consumption. The client with offered mechanism does not need to keep its connection for gathering events even if the client is behind firewall.

References

[1] Gregorik I, High Performance Browser Networking, O'Reilly Media; 2013.

[2] Fette I, The WebSocket Protocol, Internet Engineering Task Force; 2011.

[3] Stratmann E, Ousterhout J, Madan S, Integrating Long Polling with an MVC Web Framework, 2nd USENIX conference on Web application development; 2011.

[4] Castillo IB, Villegas JM, Pascual V, The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP), Internet Engineering Task Force; 2014.

[5] HTML5 Web Sockets: A Quantum Leap in Scalability for the Web, Available: http://www.websocket.org/quantum.html; 2015.

[6] Comet: Low Latency Data for the Browser, Available: https://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/; 2015.

[7] Loreto S, Saint-Andre P, Salsano S, Wilkinson G, Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP, Internet Engineering Task Force; 2011.

[8] Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, Berners-Lee T, Hypertext Transfer Protocol – HTTP/1.1, RFC2616, Network Working Group; 1999.

[9] Ravindran P, RTCWeb standard signaling protocol, RTCWeb working group; 2011.

[10] Hickman, Kipp, The SSL Protocol, Netscape Communications Corp.; 1995.

[11] Freier A, Karlton P, Kocher P, The SSL 3.0 Protocol, Netscape Communications Corp.; 1996.

[12] Rescorla E, HTTP over TLS, RFC2818, Network Working Group; 2000.

[13] Callegati F, Cerroni W, Ramilli M, Man-in-the-Middle Attack to the HTTPS Protocol, IEEE Security and Privacy, Volume 7 Issue 1; 2009

[14] Handley M, Jacobson V, Perkins C, SDP: Session Description Protocol, RFC4566, Network Working Group; 2006.

[15] Srisuresh P, Holdrege M, IP Network Address Translator (NAT) Terminology and Considerations, RFC2663, Network Working Group; 1999.

[16] Rosenberg J, Interactive Connectivity Establishment (ICE): A protocol for Network Address Translator (NAT) for Offer/Answer Protocols, RFC5245, Internet Engineering Task Force; 2010.

[17] Rosenberg J, Keranen A, Lowekamp BB, Roach AB, TCP Candidates with Interactive Connectivity Establishment (ICE), RFC6544, Internet Engineering Task Force; 2012.

[18] Doyle J, Carroll J, Network Address Translation, Cisco Press; 2002.

[19] Loreto S, Pietro Romano S, Real-Time Communications in the Web: Issues, Achievements, and Ongoing Standardization Efforts, IEEE Internet Computing, 16(5), 68-73; 2012.

[20] Uberti J, Jennings C, Javascript Session Establishment Protocol, Internet Engineering Task Force Internet draft; 2012.

[21] Eriksson G, Holmberg C, Hakansson S, Web Real-Time Communication Use-Cases and Requirements, Internet Engineering Task Force Internet draft; 2014.