Push Notifications for WebRTC clients

Dan DRUTA | AT&T

Abstract

WebRTC brings real time communications to browser based web applications. This includes voice, video and messaging as well as interoperability with existing telecommunications technologies and protocols. Along with great innovation opportunities, the new technology brings several new challenges to the end to end user experience including but not limited to security, privacy and alerting/notifications.

The goal of this paper is to:

- State the problem of push notifications in the context of WebRTC
- Outline the use cases
- Provide the state of various activities related to push
- Analyze the options
- Provide a proposal for the WebRTC WG to address the gaps

Problem Statement

Most of the web based solutions today work under the premises that parties involved are online and connected to the same web application at the same time. With the addition of phone-like capabilities to the landscape, there has to be a way to notify users of incoming requests to connect.

Therefore the WebRTC standardization efforts should take into consideration the ability of the compliant solutions to:

- Provide unsolicited notifications for incoming connection requests in WebRTC clients
- Make notifications available to the user irrespective if the app is running, in focus or not.
- Provide notifications irrespective of the browser, device type or network environment

While some use cases might have limited need for notification support, the standards set forth for WebRTC should accommodate for the scenarios that do have the need for notifications and take into consideration additional factors related to client characteristics.

WebRTC solutions are expected to connect clients using a variety of platforms including mobile end user devices. These particular devices are rather constrained in terms of the power resources and an always-on approach will have a significant impact on the battery life and as a consequence on the user experience. It is expected that several applications would attempt to maintain independent connections with server side components in order to fulfill similar requirements and scenarios. Email, news and finance applications are just a few examples of applications that make use and require notification

frameworks. Connectionless push is not just a mobile requirement. True push notifications are necessary for use cases in which the client has the ability and in the eventuality of going in stand by mode or in environments where clients are occasionally connected.

Use Cases

This section identifies two use cases that show the need for notifications in the context of WebRTC.

1. Bob uses Ringo's STAR WebRTC service on his mobile device. Bob calls Alice at work and leaves a voice mail for her to call him back. Alice comes back from her meeting and calls Bob. Bob receives an incoming notification. He can clearly identify the caller as Alice and answers.

2. Alice is using her browser on her laptop to chat with her mom overseas. She goes online and marks her status available. Then she closes her browser window to work on a presentation due next week. Her mom wants to tell her the great news about her cousin's new baby and calls her. Alice receives a notification that her mom wants to talk and answers.

Existing Developments

1. W3C Server –Sent Events – This specification defines an API for opening an HTTP connection for receiving push notifications from a server in a form of DOM events.
   The API is designed in such that it can be extended to work with other push notification schemes such Push SMS.

   Limitations:

   As informatively noted in the Connectionless section, *User Agents running in constrained environments, e.g. browsers on mobile handsets, may be capable of using additional network bearers to offload management of EventSource connections when it is advantageous, e.g. in order to conserve network and device resources (e.g. battery).*
   The EventSource API and text/event-stream wire format can be supplemented for these more distributed ways of implementing EventSource connections, with additional formats of event framing defined by other specifications, as supported.
   EventSource does not define how such additional event framing is to be parsed or processed.

2. Headless Browsers – Headless browsers are fully fledged browsers with no user interface. A headless browser can be configured as a TSR (terminate and stay resident) application to receive Server-Sent Events from the application server.
   Limitations:

While the headless browser can address the issue of application running in the background, it falls short in providing an efficient way to manage connections as each application will maintain its own connection to the server.

3. Existing platform specific notification frameworks. Mobile Operating Systems (e.g. iOS, Android, Windows Phone) have built in push notification frameworks for the native application development.
   Limitations:
   In the context of notification frameworks, the user agent is an application and while it can register to receive notifications it falls short in being able to handle notifications on behalf of web applications. To achieve this, each web application has to be wrapped in a native application.

4. EventSource API Connectionless Push Extension – Bryan Sullivan of AT&T submitted a proposal in the WebApps Working Group as an extension to the EventSource API to establish event delivery from connectionless push event sources and to use text/event-stream MIME type as a mechanism to deliver SMS and OMA Push events.

Proposal for WebRTC

This paper does not provide a concrete API proposal for the WebRTC Working Group. The proposal is to define the specific areas that have to be addressed by the group and gather consensus regarding the scope for this effort.

It is believed that WebRTC does NOT have to standardize on a protocol for push notifications.

The group should leverage previously listed developments specifically Server-Sent Events and the proposed extension for connectionless push.

WebRTC group should focus on the following areas:

- **Register a URI scheme or data format (MIME) and associate the browser as the handler**
- **Define a specific structure for the notification that would be passed**
- **Define an API to parse the structure for the notification and provide a seamless integration with the PeerConnection API**
- **API to attach to a notification service**