Overview and comparison of some libraries providing real time communication

Madis Nõmme

December 30, 2012

Abstract

Real time communication in this writing is used in the context of web applications. Traditional approach is the request-response mechanism. Real time means that information provider (server or other users of a site or service) can publish their data and it will be received by subscribers.

1 Libraries

From the following, first two(Socket.io and Faye) are open source projects. Realtime.co and Pusher are commercial services, that let developer use their infrastructure and provide client side libraries to communicate with it.

1.1 Socket.io[1]

Socket.io is a library for node.js(V8 javascript VM for server side use) that allows developers to push messages from server to clients. Out of the box it is a publish-subscribe mechanism for server and client. Both the server and the client can be in the roles of publisher and subscriber. There needs to be central server where clients can connect to.

With a bit of custom development, the server can function as a message router or hub. Thus allowing client to client messaging.

1.1.1 Features and benefits

1. Fallbacks to different transport mechanisms when WebSocket is not available
2. Namespaces (i.e. clients can connect to /administrators or /visitors)
3. Concept of rooms. A message can be broadcasted to all clients being in a room.

[1] http://socket.io/

1.2 Faye[2]

Implementation of the Bayeux protocol[3]. It is also a server-client publish-subscribe type solution. Faye is widely used in web applications written in Ruby. In these cases the web application needs to be run in a event driven web server. Currently Thin, Rainbows and Goliath web servers are supported.

1.2.1 Features and benefits

1. Adapter implementations in Ruby and Node.js
2. Easier to integrate if the web application is already implemented in Ruby(e.g. Rails or Sinatra). With socket.io there is a need to possibly duplicate the model code and other things for the web application and the messaging server.
3. Better nesting of resources where to subscribe (e.g. /messages/private/:user_session_id). Goes well together with the REST concepts.

1.3 Realtime.co[4]

A commercial product that provides publish-subscribe mechanism. The company hosts message server so developers usually only develop message clients. These can be in browser or in server. This takes away some flexibility but at the same time can give more reliability. Messaging in Realtime.co is done in terms of channels. Interested parties will subscribe to channels. They can have different permissions on channels (e.g. listen or write). There is no built in concept of rooms or namespaces.

1.3.1 Features and benefits

1. No need to have server infrastructure for messaging. Server can deal only with serving web application data.
2. Scales well when there are many clients and many messages (millions of messages per second).
3. Client to client(browser to browser) messaging out of the box.
4. Library implementations for multiple platforms and languages (Node, .NET, Ruby, etc.).

1.4 Pusher[5]

Also a commercial service. Provides similar functionality as the Realtime.co. Internally uses WebSockets and has fallbacks to Flash socket.

[2] http://faye.jcoglan.com/
[3] http://svn.cometd.com/trunk/bayeux/bayeux.html
[4] http://www.realtime.co/
[5] http://pusher.com/

1.4.1 Features and benefits

Mostly the same as in Realtime.co. But in addition to these:

1. Rooms and namespaces are also supported.
2. Provides REST API for communication between server webapp to Pusher service.

2 Conclusion

These four solutions all serve one purpose - to enable pushing state from server to browsers (or other consumers of the service). First two are open source, latter two are commercial solutions. Commercial solutions can have the benefit of scaling better and providing a bit more out-of-the-box functionality.

I have implemented solutions in Faye and Socket.io and am proponent of open source development. The open source solutions have more flexibility in configuring and setting them to suite one's specific needs. They also often adopt faster to new changes in the browser-server wild evolution. Most web applications don't need to be in massive scale. Serving few hundred to thousand simultanious connections with socket.io or Faye is not a problem on current hardware. The bottlenecks are usually in the business logic and database components. So developers need to make components serving these areas to scale rather than the message pushing component.

	socket.io	Faye	Realtime.co	Pusher
Free	yes	yes	Free up to 30k visits or 1M messages per month. Different plans after that.	Free for 20 connections and 100k connections per day. Different plans after that.
Open source	yes	yes	no	no
Transports	WebSocket with fallbacks(web-socket, htmlfile, xhr-polling, jsonp-polling, flash socket)	WebSocket implementation in Ruby. Can use different adapters including socket.io which provides all socket.io fallbacks.	WebSocket with polling fallback.	WebSocket with polling fallback.
Standardized	No but source code is easily readable. Uses common node.js idioms (.on and .emit methods and callbacks)	Yes. Implements the Bayeux protocol.	No	No
Platforms, languages	Javascript	Ruby, Javascript	Bindings for Javascript, PHP, ASP.net, Java	Javascript
Size(browser side client library)	21kB	30kB	Initially 263kB and lazy-loads in additional files.	30k
Security	Yes, SSL	Yes, SSL	SSL + channel based permissions.	SSL only in paid versions

Table 1: Comparison