Comparative Analysis of Performance Using Server-Client Protocols

Mihail Costea
University Politehnica of Bucharest
The Faculty of Automatic Control and Computers
Bucharest, Romania
Email: mihail.costea90@gmail.com

Liviu Chircu
University Politehnica of Bucharest
The Faculty of Automatic Control and Computers
Bucharest, Romania
Email: liviu.chircu@gmail.com

***Abstract*—TODO - add references**

Current web applications' solutions for bi-directional communication are based on AJAX. Even though they are well documented solutions that are backed up by years of utilization, they have limitations imposed by the HTTP protocol. HTTP is a stateless protocol that requires each connection to be treated as a new connection, requiring an unnecessary overhead to communicate in both directions. Because of these limitations, a new solution was developed - WebSockets - which are able to natively mantain a bi-directional channel, thus reducing the overhead imposed by connection establishment.

This paper proposes to exemplify the advantages and disadvantages between traditional HTTP implementations for bi-directional communication based on AJAX and WebSockets. It also proposes an architecture for a testing platform for different WebSockets implementations.

I. Introduction

Along with the introduction of Web 2.0, web applications have been able to modify the content of HTML documents without the need of a complete page refresh, offering a more interactive environment for the end-user. The most popular technology used to create this interactivity is AJAX (Asynchronous JavaScript and XML). With AJAX, pages can be updated in real-time, without needing an explicit action from the user. But because AJAX is based on HTTP - a stateless protocol that requires each connection to be treated as a new connection - bi-directional channels between 2 devices can only be simulated through a series of methods which impose a significant amount of overhead, as both nodes need to mimic this channel that require states. Bi-directional channels are necessary for web applications such as instant messaging clients, browser-based games, video calls, etc., where large amounts of HTTP traffic (with complete sets of headers and message bodies) are sent from each endpoint over to the opposite one.

The most popular solutions used by AJAX are: **polling**, where a client sends a request to a server at regular intervals with the server responding immediately and closing the connection; **long-polling**, where the client requests information from the server just as in normal polling, but the server now maintains its connection open until it has the data to send, point in time where it will send an immediate response; **streaming**, where the connection between the client and the server is kept alive indefinitely and data is streamed until one of the two closes the connection. The problem with last solution is that AJAX appends the new data to previously sent data until the connection ends, which is unnecessary in many cases. Moreover, HTTP headers must be sent whenever a new connection is created, which is a common case for polling and long-polling. If a great amount of small pieces of data are to be sent, as in the case of a web-based chat application, the overhead of the headers might actually overweigh the data that is actually transmitted(**add reference in bibliography [1]**).

In order to overcome the limitations of AJAX-based solutions, the HTTP-based WebSocket protocol was created (**add RFC reference in bibliography [2]**). HTTP was chosen as a base because most firewalls allow HTTP and HTTPS traffic (ports 80 and 443), while other ports' traffic might be blocked. Although based on HTTP, WebSocket does not inherit its limitations. HTTP is only used to create and close the connection between client and server, while the in-between information exchage is done through a series of WebSocket-specific data frames. The distinction between normal HTTP traffic and WebSockets is done through the "Upgrade: websocket"/"Connection: Upgrade" headers (**RFC reference [2]**).

As mentioned above, WebSocket is a recently created technology (the RFC was written in 2011), so it has not been thoroughly tested. This may very well be considered a downside when compared to AJAX, which benefits from years of testing and industry usage. Although a series of testing platforms with the purpose of validating WebSocket implementations have been created, (**give a few examples**), to our best knowledge, we have yet to find any performance-oriented testing platforms for a given WebSocket implementation. This paper compares AJAX with WebSocket and proposes an architecture for a testing platform, which should be independent of the underlying Operating System and present devices. It should work for both laptops and smartphones, for Windows and Linux and other devices and Operating Systems.

II. Related work

Real-time Monitoring using AJAX and WebSockets paper (**create reference**) exemplifies the differences in terms of performance between AJAX and WebSockets. The authors added real-time monitoring support for OASIS (**maybe ref-**

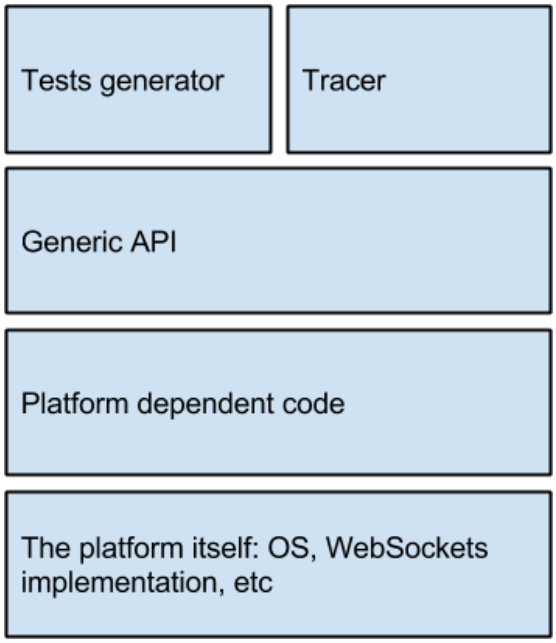

Fig. 1. Performance application architecture

erence), an open-source real-time instrumentation middleware for distributed real-time and embedded systems. The collected instrumentation data was sent over the Web using AJAX and WebSockets and the results were compared. The WebSockets server consumes 50% less network bandwidth then the AJAX server. Also, the WebSocket client consumes memory at a constant rate, while the AJAX client consumes it at an increasing rate. Furthermore, the Websocket-oriented implementation can send up to 215.44% more data samples than the AJAX-based counterpart although demanding the same amount of network bandwidth.

Another example where WebSockets are better than AJAX is the case of sending small data per frame. While WebSockets exchange 2B of data per frame, continuous polling with AJAX exchange up to 8 KB of HTTP header (**create reference**).

Even though WebSockets are better than AJAX based solutions, it's still not as good as raw TCP sockets. *Real-time Web Application Roadblock: Performance Penalty of HTML Sockets* paper (**create reference**) discusses the penalties of using HTML socket streams (long-polling and WebSockets) versus TCP streams. HTML socket streams can have up to 5x protocol overhead, up to 3x more payload delivery delay and up to 3x less throughput. In case of small data payload (a few hundreds of bytes), the performance between the two becomes significantly noticeable. Also TCP streams tend to behave better over 3G than HTML socket streams. One reason for the poorer performance is due to the fact that the browser uses buffering for HTML socket streams (both long-polling and WebSockets), thus introducing delays, while TCP sockets are able to send the data directly. But on the good side for WebSockets, the paper mentions that they behave better than long-polling when it comes to sending small-sized chunks, making it a better choice for chat, VoIP, online games, etc, just like TCP streams. A big plus for HTML sockets over TCP streams is the fact that the data sent through them can pass over firewalls and most proxies, as HTTP and HTTPS are commonly allowed ports.

III. Architecture

This section proposes a simple architecture for an application which is capable of testing the performance of WebSockets and AJAX implementations of bi-directional communication. Figure 1 contains the architecture and the layers. In order to test the performance automatically, the application should be able to generate tests based on a given input and automatically measure how those tests behave on a given platform. The design should promote easy porting to other devices and Operating Systems. It should have a layer which is independent of platform and implementation (either WebSockets or AJAX). The layer provides a generic API that is used by the tests generator component and the tracer or measurements component. The API and the previously mentioned components are going to be ported to other platforms with minimal changes. The layer that provides the API is going to be implemented by every platform in a different way, depending on the Operating System, specific WebSockets API requirements, etc.

References

[1] Frank Greco Peter Lubbers. Html5 web sockets: A quantum leap in scalability for the web. -1, 2010.
[2] Nikolai Qveflander. Pushing real time data usinghtml5 web sockets. 2010.