Asynchronous Chat: Comparing REST-ful and Protocol Buffer Implementations

Serena Booth, Michelle Cone, Nicholas Mahlangu, Tianyu Liu

March 2, 2016

1 Hint from Waldo on how to write paper

2 Introduction

REST-ful communication protocols have taken the web by storm. In this design specification, we compare this exceedingly popular communication protocol with alternate protocol buffers, having created a web application to facilitate online chat between users and groups using each of these communication protocols.

2.1 Interfaces Being Looked At

2.2 Assumed Environment

3 REST-ful Design

3.1 Long Polling

In implementing our web chat application using the REST-ful communication protocol, we opted for a long polling approach, wherein the client opens a webpage which initiates frequent, constant communication with the server by means of a series of GET requests.

We extend the long-polling approach to include a success handshake so as to confirm message receipt. In this way, if a message send operation is unsuccessful, we retry sending the message.

This long polling approach operates as follows:

- The client opens a webpage which initiates a GET request using AJAX. This GET request is open until one of the following occur: (a) a success response is received or (b) an error response is received.

- In either (a) or (b), a completion script runs in which the AJAX request is dispatched again after a set amount of time. In the case of (a), there is no delay; in the case of (b), there is a 1000 millisecond delay.
- The server receives the GET request. It looks up messages for the particular client, based on a cookie passed in the request header, and sends an unread message back to the client, along with a success response. Further, the server sends the ids of the message returned in the response header. Lastly, the server updates the MySQL database of messages to indicate that this message is currently being sent.
- On (a), the success response, the client receives a message to display and an id corresponding to that message from the MySQL database. The client then dispatches an additional GET request containing the id of the message.
- On receiving the second confirmation GET request, the server updates the MySQL database to indicate that the message corresponding to a particular id has been received.
- If the server does not receive a success response a minute after sending a message, on the next client-initiated GET message corresponding to a request for new messages, the server updates the MySQL database to indicate that the formerly sent message was not received.

3.2 Resources “Conserved”

3.3 Resources “Wasted”

3.4 Failure Conditions

The use of a REST-ful communication protocol lends this distributed system to failstop, crash, receive-omission, send-omission, and general omission failures.

4 Protocol Buffer Design

4.1 Resources “Conserved”

4.2 Resources “Wasted”

4.3 Failure Conditions

5 Approach Comparison

6 Conclusion