ANEXD

James Grant
Fred Barnes
CO600: Communication Protocol Research

Communication Protocol Research

Expectations

As defined by our requirements, we needed to find an optimal way for connections to our platform to operate. Most real time multiplayer applications, such as video games communicate via a TCP connection to a server. However, our requirement is to communicate through a web application in a browser. Some research quickly lead us to WebSocket; a protocol widely supported in most modern browsers, and is a standardised way of communicating over an underlying TCP connection on the web, while offering additional security features such as a handshake.

We found that most major programming languages offered support for WebSocket connections, and did some research on how they performed when stress tested by WebSocket connections.

Benchmarks

The following test exhibits 10000 users connecting to and then sending a message to the server over 5 minutes, and measures the time taken for a response:

Implementation	Handshake Time (σ)	Latency (σ)	Connection Timeouts
erlang-cowboy	101.107	596.273	0
go-gonet	140.531	1137.411	0
java-webbit	123.325	453.46	1
python-gevent-websocket-N	829.23	13912.214	10
haskell-snap	962.598	39791.393	494
node-ws	1716.92	70812.913	1425
perl-ev	4027.54	2025.464	4371

Source: http://eric.themoritzfamily.com/websocket-demo-results-v2.html

As these results found, Erlang, Go and Java were the only 3 languages that handled this many connections effectively. The others; Python, Haskell, Node.JS and Perl all had significantly high response latency under this load, and many connections failed entirely.

While Java performed well in response times, it did drop a connection. Erlang seemed to handle connections the best and most consistently, while Go was the only other language to not drop any connections, with decent performance too.

Of course, other factors would need to be taken into account in selecting a suitable language for implementing a server for Anexd, such as library support for common libraries, and overall concurrent processing performance.

WebSocket Libraries

There are many WebSocket implementations in existence. Two of the most popular are 'Socket.IO' and 'Sock.JS'.

Socket.IO has many features going for it:

- Rooms and Namespaces system to group connected users.
- Stores client data within instances.
- Handles connection transparently; including upgrading/downgrading to the best available connection protocol.
- Scalable over multiple server nodes.
- Simple and streamlined API.
- Has a JSON serialiser built in – converting data sent into the language's respective data type.
- Wide language support.

Sock.JS advantages:

- Scalable over multiple server nodes.
- More consistent behaviour between client and server implementations.
- Wide language support.
- More simple implementation closer to standard WebSocket.

After researching the two, we decided that **Socket.IO** would be the API we would use for implementing Anexd, as we felt that the additional functionality that it provides would increase the potential of the platform. However, we found that Erlang did not have a library for Socket.IO 1.0+. For this reason, we chose Go as our choice programming language, as it has an up to date Socket.IO implementation by 'googollee':

Source: https://github.com/googollee/go-socket.io