Websocket protocol RFC 6455 implementation description

http://tools.ietf.org/html/rfc6455

Yong Jae Shin 11353644.

Websocket is the latest web technology to improve real time communications in web environments instead of using standard HTTP. HTTP requires heavy request / response headers in order to communicate with clients so it is not an ideal technology for real time applications. On the other hand, Websocket was introduced to provide light weight data communication between server and clients, furthermore, it provides full duplex communications over TCP so it requires no request and response operations in order to send and receive data. Expressive power of Bondi enabled straightforward implementation of the protocol. Unique features of the language including path polymorphism and function extension helped to build a program more efficiently than in other languages.

Project goal

This project is about creating a Websocket API that runs on Bondi language and it can be used by other peers who are interested in this technology as well. In addition, the goal is to simplify an algorithm using Bondi's expressive power, unique features and Multi-paradigm language feature.

How Websocket protocol works

1. Handshaking

Websocket protocol resembles HTTP because it requires handshaking between client and server. When client connects to the server, the client immediately sends a handshake request data that looks as follow

GET / HTTP/1.1
Host: localhost
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: "base64 encoded key here"

Then the server parses this data and retrieves necessary data. Sec-WebSocket-Key is a password that needs to sent back to a client by concatenating new data and encodings. The server then responds to the client by sending following information

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: "generated key here"

2. **Framing**

Once the handshake negotiation is completed and both sides agrees to continue forward is they start to send data encoded in frames. Framing is used by both peers to encode data before sending it to each others. Frames contain unsigned bytes data. First byte represents header information of this frame.

FIN (Tells this is a frame, 1 bit)

rsv1, rsv2, rsv3 (Used for extensions, 3 bits)

opcode (Type of actual information, text or bytes or int, 4bits. 0001 represents text)

In the project, a byte of frame looks like 1000 0001 because it is a FIN, uses no extensions and it contains text data. Then the frame contains bytes for length and actual data.

Bondi features in this project

1. **Path polymorphism**

Path polymorphism simplifies data sending operations.

```
%hide types;;
let sendText text client =
let bytes = getBytes text in
let rec sendBytes =
| (x:String)     -> write client x
| (x:Int)        -> writeint client x
| Cons x xs      -> sendBytes x; sendBytes xs in
sendBytes bytes;
sendBytes text;;
~
```

In order to send a new data to WebSocket protocol receiver, each data must be encoded into frames which is mentioned in the RFC reference. In Java implementation or many other languages, they would use getBytes function on Strings

"*Hello websocket client".getBytes("UTF-8");*

to get bytes array and join it with the original bytes array. In Bondi, it simply puts the raw "Hello websocket client" string inside the original array and use path polymorphism to send them appropriately (write String->Unit and writeint int ->Unit) according to their types.

Example array [129, 12,"Hello client"]

2. Function extension

Websocket Protocol requires to go through a handshaking negotiation like HTTP. Function extension feature of Bondi offers much easier way to implement this process. First of all, verifyRequest function parses data from incoming request messages and extend them to a pattern matching function getRequest.

```
let ext getRequest =
| _ -> "Un";;

let rec verifyRequest =
| (x:String) ->
      if contains x methodz
              then getRequest += | "method" -> (getSecond x del)

       else if contains x connection
              then getRequest += | "connection" -> (getSecond x del)
       else if contains x key
              then getRequest += | "key" -> (getSecond x del)
       else if contains x upgradeValue
              then getRequest += | "upgrade" -> (getSecond x del)
       else ();;
```

Using the same getRequest function, validation process becomes much simpler. In a typical Object Oriented language, this process must have done using a class called Request with list of attributes for each piece of data.

```
 if (proc client 0) == 1 then

        if (getRequest "key") == "Un"
                then Exception "Client did not send a socket

        else writeInfo ("RequestValid", "socket key checked")

        if (getRequest "method") == "Un"
                then Exception "Request method in invalid"
        else writeInfo ("RequestValid", "Method checked");

        if (getRequest "connection") == "Un"
                then Exception "Unknown connection valud rece

        else writeInfo ("RequestValid", "connection verified"


        if (getRequest "upgrade")  == "Un"
                then Exception "Upgrade unknown"
        else writeInfo ("RequestValid", "Upgrade value verifi

 else Exception "Could not verify client request!"
```

Interesting features in the project

1. **Overriding toString to Websocket class**

This feature greatly helps users who are going to use this API. Once a new Websocket object is created, it tells Websocket version and how to find RFC specification online.

```
with toString += | (x:WebSocket) -> "Websocket protocol for version 13\n "
                 ^ " host: "^ (!x.hostz)
                 ^ "  port: " ^ (toString (!x.port))
                 ^ "\nWebsocket version " ^ (toString wsVersion)
                 ^ "\nPlease Google 'Websocket RFC version 13' for protocol specificat
```

2. **Logger library**

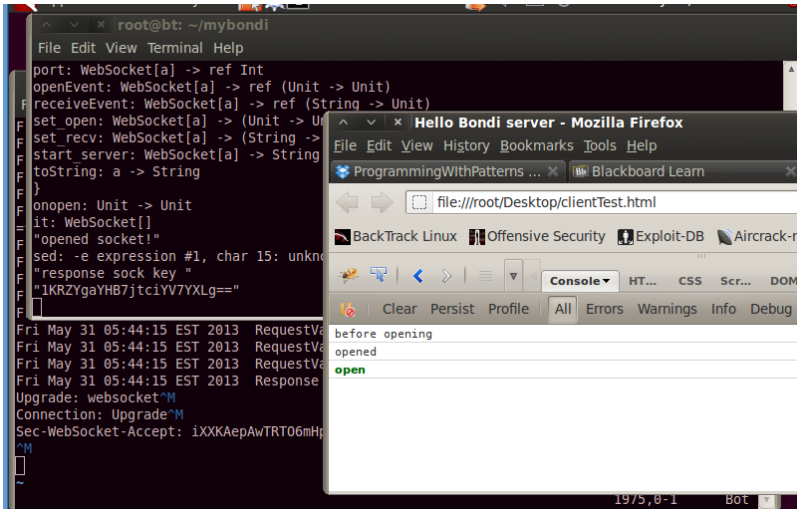
Logger library is used to record the program's process. It has different logging levels, including warning, error, verbose and information. For instance, writeInfo ("Request", "test") will save a following string line to a logging file.

Fri May 31 05:44:15 EST 2013 Request INFO test

This helps to see past results and focus on error or warning level logs to debug errors.

Project progress

Currently client successfully accepts handshakes and allows to start sending frames.



However, although the project currently has Framing implementation, it does not allow me to send unsigned bytes (0~255) data which is essetial for clients. The intention of project was to implement a protocol using Bondi's powerful features.