WSGI, GEvent and Web Sockets

Finnbarr P. Murphy

(fpm@fpmurphy.com)

Web Sockets is an emerging technology that enables bi-directional, full-duplex communications channels, over a single TCP socket. It was originally designed to be implemented in web browsers and web servers, but it can be used by any client or server application. Tests have demonstrated Web Sockets can provide a 500:1 reduction in network traffic and 3:1 reduction in latency.

Web Sockets have been around in one form or another since 2009. However specifications are still not yet fully cooked. The W3C Web Applications Group is responsible for standardizing the WebSocket API. The editor of the W3C TR is Ian Hickson who is generally regarded as the inventor of Web Sockets. On the other hand, the WebSocket protocol is still evolving and is being standardized by the IETF HyBi (Hypertext-Bidirectional) Working Group. The editor is Ian Fette of Google Inc. By the way, agreement on the technology has not been smooth, there have been over 80 drafts of the specifications so far.

Based on feedback from early implementers, the WebSocket protocol was undated in 2010 to *draft-ietf-hybi-thewebsocketprotocol-00* (also known as *draft-hixie-thewebsocketprotocol-76*). This version relaxed requirements on the handshake to make it easier to implement with HTTP libraries, and introduced nonce-based challenge-response to protect from cross protocol attacks. However these changes made the protocol incompatible with the previous version, i.e. *draft-hixie-thewebsocketprotocol-75*. Thus a *-75* protocol implementation does not work with a *-76* protocol implementation.

Here is the formal definition of the WebSocket API:

```
[Constructor(in DOMString url, optional in DOMString protocol)]
interface WebSocket {
readonly attribute DOMString URL;
// ready state
const unsigned short CONNECTING = 0;
const unsigned short OPEN = 1;
const unsigned short CLOSED = 2;
readonly attribute unsigned short readyState;
readonly attribute unsigned long bufferedAmount;

// networking
attribute Function onopen;
attribute Function onmessage;
attribute Function onclose;
boolean send(in DOMString data);
void close();
};
WebSocket implements EventTarget;
```

We will use JavaScript in the following examples but there is no reason other languages such as Python cannot be used with Web Sockets.

To connect to a WebSocket end-point, create a new WebSocket instance as shown in the following example. All you need to provide is a URL that represents the end-point to which you wish to connect.

```
var myWS = new WebSocket("ws://www.fpmurphy.com");
```


The WebSocket protocol specifies the *ws://* prefix to indicate a WebSocket and the *wss://* prefix to indicate a Secure (TLS) WebSocket connection.

A WebSocket connection is established by an upgrade from the HTTP protocol to the Web Sockets protocol during the initial handshake between a client and a server, over the same underlying TCP/IP connection. Once established, WebSocket data frames can be sent back and forth between the client and the server in full-duplex mode.

Before connecting to an end-point, you can associate event listeners to handle each phase of the connection life-cycle as shown in the following example:

```
myWS.onopen = function(evt) { alert("Connection open ..."); };
myWS.onmessage = function(evt) { alert( "Received Message: " + evt.data); };
myWS.onclose = function(evt) { alert("Connection closed."); };
```

At a minimum, you need an *onmessage* event handler if you wish to receive messages.

To send a message, use the *send* method.

```
myWS.send("Hello World!");
```

To close the connection:

```
 myWS.close();
```

Turning now to gevent which is a coroutine-based Python networking library that uses greenlet to provide a high-level synchronous API on top of the libevent or more recently the libev event loop. Gevent is generally regarded as one of the best performing asynchronous frameworks out there at present. It includes two WSGI servers, i.e. *wsgi.WSGIServer* and *pywsgi.WSGIServer*.

Jeffrey Gelens has developed a WebSocket library for the *gevent* networking library. In the following example we use this library to develop a simple sample of a WebSocket enabled webserver which responds to a message from a client, sends back 100 randomly generated numbers between 0 and 1 which, in turn are displayed on a realtime graph by the client. Most of this code is not mine, I simply pulled it together into this example.

Here is the server code:

```
#!/usr/bin/python
import os
import sys
import random
import threading
import webbrowser
from geventwebsocket.handler import WebSocketHandler
from gevent import pywsgi
import gevent
FILE = 'test.html'
PORT = 8000
def handle(ws):
    if ws.path == '/echo':
        while True:
            m = ws.wait()
```



```
            if m is None:
                break
            ws.send(m)
    elif ws.path == '/data':
        for i in xrange(100):
            ws.send("0 %s %s\n" % (i, random.random()))
            gevent.sleep(0.1)
def app(environ, start_response):
    if environ['PATH_INFO'] == '/test':
        start_response("200 OK", [('Content-Type', 'text/plain')])
        return ["Yes this is a test!"]
    elif environ['PATH_INFO'] == "/data":
        handle(environ['wsgi.websocket'])
    else:
        response_body = open(FILE).read()
        status = '200 OK'
        headers = [('Content-type', 'text/html'), ('Content-Length', str(len(response_body)
))]
        start_response(status, headers)
        return [response_body]
def start_server():
    print 'Serving on http://127.0.0.1:%s' % (PORT)
    server = pywsgi.WSGIServer(('0.0.0.0', PORT), app,
             handler_class=WebSocketHandler)
    server.serve_forever()
def start_browser():
    def _open_browser():
        wb = webbrowser.get('/usr/bin/google-chrome %s')
        wb.open_new_tab('http://localhost:%s/%s' % (PORT, FILE))
    thread = threading.Timer(0.5, _open_browser)
    thread.start()
if __name__ == "__main__":
    print "version:", sys.version
    start_browser()
    start_server()
```

When the server is started, it first starts an instance of the Google Chrome browser, or a new tab if an instance of the Google Chrome browser is already running, in a separate thread of execution. The browser then displays the following webpage (*test.html*) to the user.

```


<!DOCTYPE html>
<html>
    <head>
        <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.1/jquery.min.js"></sc
ript>
        <script src="http://people.iola.dk/olau/flot/jquery.flot.js"></script>
        <script>
            $(document).ready(function() {
                 if ( typeof(WebSocket) != "function" ) {
                     $('body').html("<h2><center>ERROR: This browser does not support Web
Sockets</center></h2>");
                 }
            });
            function plotnumbers() {
                var data = {};
                var s = new WebSocket("ws://localhost:8000/data");
                s.onopen = function() {
                    s.send('hi');
                };
                s.onmessage = function(e) {
                    var lines = e.data.split('\n');
                    for (var i = 0; i < lines.length - 1; i++) {
                        var parts = lines[i].split(' ');
                        var d = parts[0], x = parseFloat(parts[1]), y = parseFloat(parts[2
]);
```



```
                        if (!(d in data)) data[d] = [];
                        data[d].push([x,y]);
                    }
                    var plots = [];
                    for (var d in data) plots.push( { data: data[d].slice(data[d].length
- 200) } );
                    $.plot( $("#placeholder"), plots,
                    {
                        series: {
                            color: "rgba(255, 0, 0, 0.8)",
                            lines: { show: true,
                                     lineWidth: 5,
                                     fill: true,
                                     fillColor: "rgba(255, 162, 0, 0.6)"
                            },
                         },
                         yaxis: { min: 0 },
                    } );
                    s.send('');
                };
            };
        </script>
    </head>
    <body>
         <center><h3>Plot of Random Numbers</h3><center>
         <div id="placeholder" style="width:600px;height:300px"></div>
         <br />
         <input type="submit" class="button" value="Start Plot" onClick="plotnumbers();" /
>
    </body>
</html>
```

When the user presses the *Plot* button, a WebSocket is created between the browser and the server and 100 random numbers between 0 and 100 are sent to the browser. Here, the numbers are processed into a format that *flot*, a popular JavaScript plotting library for jQuery, understands.

If you examine the above code, you will see that it contains two JavaScript functions. The *onload* function checks to ensure that you are using a WebSocket-enabled web browser. If not, a simple error message is displayed and nothing else occurs. The *plotnumbers* function does all the real work. It creates a WebSocket connection to the server and processes the returned numbers in the *s.onmessage* event handler. Most of the code here relates to preparing the numbers for display by *flot* (see *$.plot($("#placeholder"))*.

Here is a video of the above example in action. This does not display in the current version of Firefox. You need to use a WebM-based browser such as Google Chrome.

http://blog.fpmurphy.com/video/gevent1.webm

By the way, the video was made using the build-in *WebM* video recorder in the Gnome Shell and the videoplayer is the excellent mediaplayer.js HTML5 player which can also provide a non-HTML5 fallback mode (which I have not enabled).

Web Sockets are going to change the way web-based applications are developed and deployed. For example, SocketStream which is an emerging real-time full stack Single-page Application (SPA) framework for Node.js makes extensive use of Web Sockets.

You can download a tarball of this example here. Just download, unpack and invoke *test.py* to use it. You may see a *Failed to create ~/.pki/nssdb directory.* error message. This is a bug in Chrome.

