



Inspecting WebSocket Traffic

When experimenting and building applications with WebSockets, occasionally you may need to take a closer look at what exactly is happening under the covers. Throughout this book, we've used some of these tools to examine WebSocket traffic. In this appendix, we review three handy tools:

- Google Chrome Developer Tools: a set of HTML5 applications that ships with Chrome and allows you to inspect, debug, and optimize Web applications
- Google Chrome Network Internals (or “net-internals”): a set of tools that allows you to inspect network behavior including DNS lookups, SPDY, HTTP caching, as well as WebSocket
- Wireshark: a tool that enables you to analyze network protocol traffic

WebSocket Frame Inspection with Google Chrome Developer Tools

Google Chrome Developer Tools offer a wide range of features to help web developers. Here we focus on how it helps you learn about and debug WebSockets. If you're interested in learning more about Google Chrome Developer Tools in general, there's plenty of information available online.

To access the Developer Tools, open Google Chrome, then click the Customize and Control Google Chrome icon, located to the right of the address bar. Select **Tools ► Developer Tools**, as shown in Figure A-1. Most developers who use this tool frequently prefer the keyboard shortcut to the menu selection.

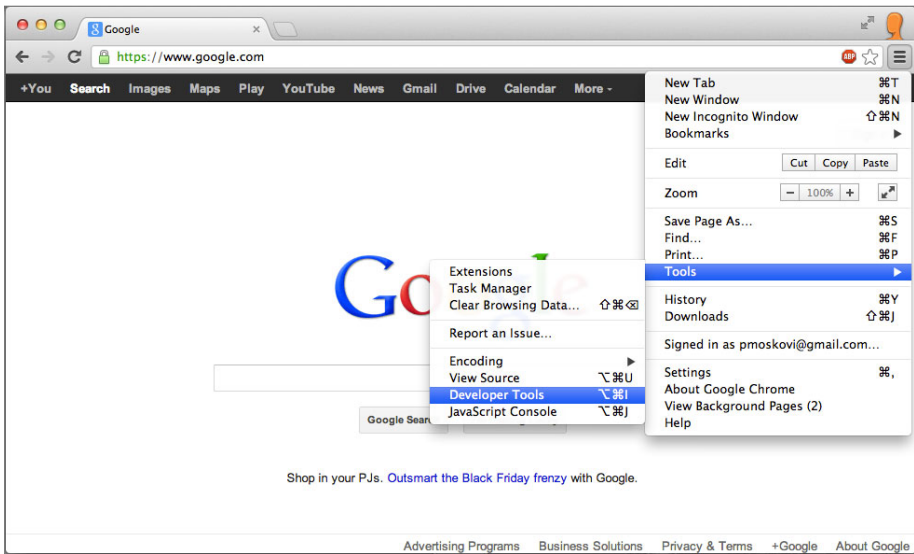


Figure A-1. Opening Google Chrome Developer Tools

Google Chrome Developer Tools provide you with detailed information about your page or application through eight panels, allowing you to perform the following tasks:

- Elements panel: inspect and modify the DOM tree
- Resources panel: inspect resources loaded
- Network panel: inspect network communication; this is the panel you'll use the most while building WebSocket-enabled applications.
- Sources panel: inspect source files and debug JavaScript
- Timeline panel: analyze where time is spent when loading or interacting with your page
- Profiles panel: profile the time and memory usage
- Audits panel: analyze the page as it loads and makes suggestions to improve it.
- Console: display error messages and execute commands. The console can be used along with any of the above panels. Press the Esc key on your keyboard to open and close the console. Along with the Network panel, the Console is the Web and WebSocket developer's best friend.

First, let's take a closer look at the Network panel. Open Chrome and navigate to <http://www.websocket.org>. We will use the Echo Test on [websocket.org](http://www.websocket.org) to learn about

the WebSocket Frame inspection that Google Chrome Developer Tools provide. To access the Echo demo, click the Echo Test link on the page, which will take you to <http://www.websocket.org/echo.html>. Open Google Chrome Developer Tools if you haven't opened it yet, and click the Network panel. Make sure your Network panel is empty. If it is not empty, click the Clean icon at the bottom of the Chrome Window, the sixth icon from the left in Figure A-2.

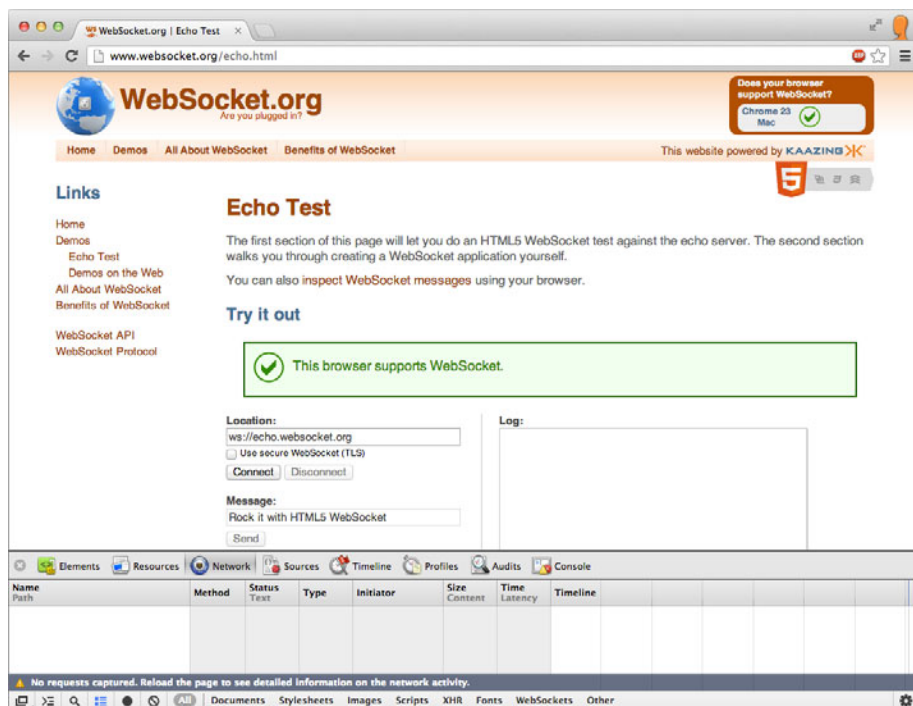


Figure A-2. Examining the creation of a WebSocket connection with Google Chrome Developer Tools

Notice that the location field contains a WebSocket URL that we'll connect to: `ws://echo.websocket.org`. Click the Connect button to create the connection. Notice that the WebSocket connection displays in your Network panel. Click the name, `echo.websocket.org`, which is under the Headers tab; doing so allows you to look at the WebSocket handshake (Figure A-3). Listing A-1 shows the entire WebSocket handshake.

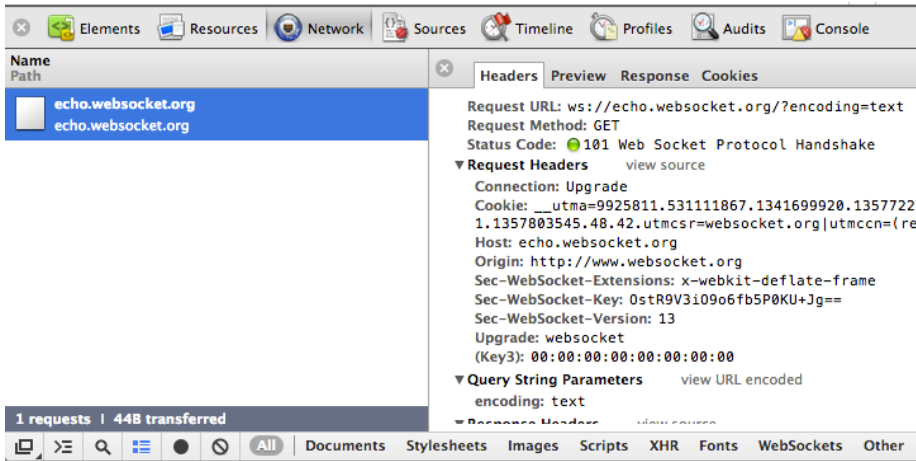


Figure A-3. *Inspecting the WebSocket handshake*

Listing A-1. The WebSocket Handshake

Request URL: <ws://echo.websocket.org/?encoding=text>

Request Method:GET

Status Code:101 Web Socket Protocol Handshake

Request Headers

Connection:Upgrade

Cookie: __utma=9925811.531111867.1341699920.1353720500.135372

5565.33; __utmb=9925811.4.10.1353725565; __utmc=9925811; __
utmoz=9925811.1353725565.33.30.utmcsr=websocket.org|utmccn=(referral)|

utmcmd=referral|utmctt=/

Host: echo.websocket.org

Origin: <http://www.websocket.org>

Sec-WebSocket-Extensions:x-webkit-deflate-frame

Sec-WebSocket-Key: JfyxfhR8QIm3BSb0q/Tw5w==

Sec-WebSocket-Version:13

Upgrade:websocket

```
(Key3):00:00:00:00:00:00:00:00
```

Query String Parameters

```
encoding:text
```

Response Headers

```
Access-Control-Allow-Credentials:true
```

Access-Control-Allow-Headers:content-type

Access-Control-Allow-Origin:<http://www.websocket.org>

Connection:Upgrade

Date: Sat, 24 Nov 2012 03:08:27 GMT

Sec-WebSocket-Accept:Yr3WGnQMtP0ktDVP1aBU3l5DfFI=

Server:Kaazing Gateway

Upgrade:WebSocket

```
(Challenge Response):00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
```

Now, feel free to change the contents of the Message field and click the Send button. To inspect the WebSocket frames, you'll need to click on the Name on the far left again, which will refresh the panel on the right, adding the Frames tab, as shown in Figure A-4.

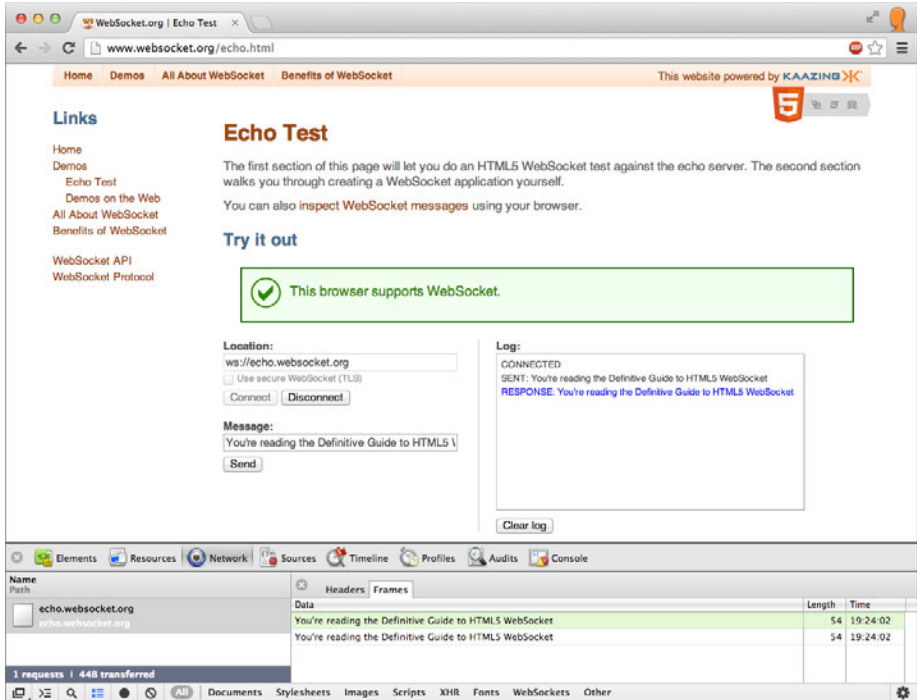


Figure A-4. Inspecting WebSocket frames

The WebSocket Frame inspector shows the data (which is text in this example), the length of the data, the time it was sent, as well as the direction of the data: a light green background indicates traffic from the browser to the WebSocket server (upload), and white indicates traffic from the server to the browser (download).

■ **Note** As you're sending WebSocket messages, be sure to always click the Name column to trigger the refresh of the Frames tab.

As you navigate to the Sources tab, and locate the `echo.js` file, you see a variable called "websocket" that represents our WebSocket connection. By displaying the Console, you can simply send a message to the WebSocket server, using the `send()` function, as shown in Listing A-2.

Listing A-2. Sending a WebSocket Message Using the Chrome Console

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

In Figure A-5 we sent a Hello World! message from the console, and you can see that in the Log window, the Echo service sent us a response. If you display your Network tab, you can also see the corresponding WebSocket frames.

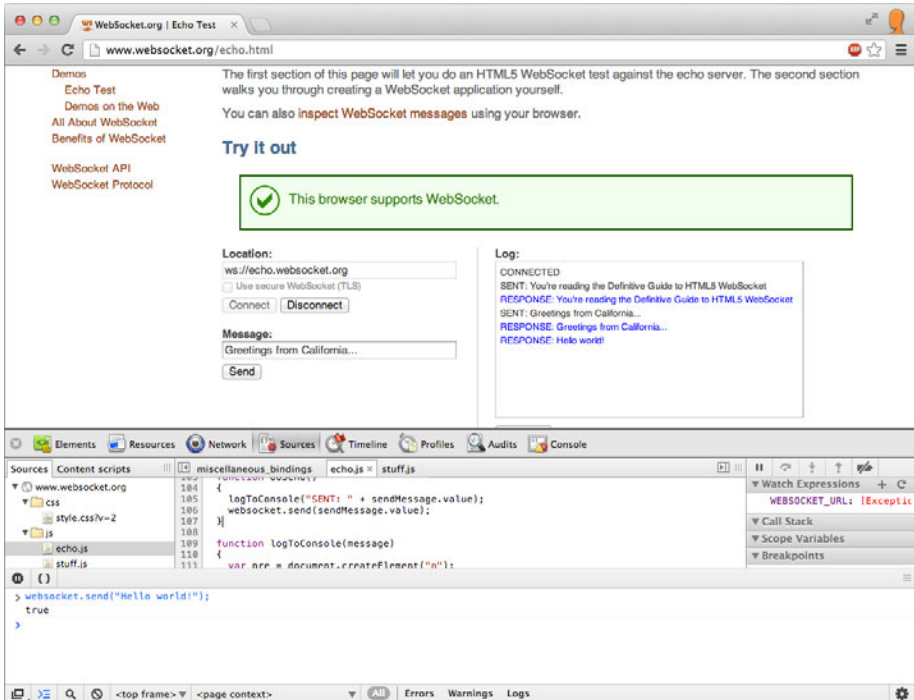


Figure A-5. Sending WebSocket messages from the Chrome Console

As demonstrated, the Chrome Developer Tools offer web developers a simple and effective way to “look under the hood” of their applications. Chrome’s Network tab provides unique insight not only into the WebSocket handshake but also allows you to easily inspect the WebSocket frames.

Google Chrome Network Internals

Most of the time, Chrome Developer Tools display more than enough information to productively develop and debug web applications. Sometimes, however, lower-level details can help diagnose unusual connection failures or provide otherwise inaccessible information when investigating the behavior of the browser itself. Chrome has internal diagnostic pages that are extremely valuable in those rare situations in which you would

like to observe the internal state of the browser. Chrome's internal tools expose events related to DNS requests, SPDY sessions, TCP timeouts, proxies, and other internal workings of the browser.

Google Chrome includes several additional utilities. For a list of them, type `chrome://about` in the browser's address bar.

■ **Note** In Google Chrome, the URL `about:about` redirects to `chrome://about`. Other browsers, such as Mozilla Firefox, have useful URLs listed on their `about:about` pages.

The page displays the following list of useful internal Chrome utilities:

- `chrome://appcache-internals`
- `chrome://blob-internals`
- `chrome://bookmarks`
- `chrome://cache`
- `chrome://chrome-urls`
- `chrome://crashes`
- `chrome://credits`
- `chrome://dns`
- `chrome://downloads`
- `chrome://extensions`
- `chrome://flags`
- `chrome://flash`
- `chrome://gpu-internals`
- `chrome://history`
- `chrome://ipc`
- `chrome://inspect`
- `chrome://media-internals`
- `chrome://memory`
- `chrome://nacl`
- `chrome://net-internals`
- `chrome://view-http-cache`
- `chrome://newtab`

- `chrome://omnibox`
- `chrome://plugins`
- `chrome://policy`
- `chrome://predictors`
- `chrome://profiler`
- `chrome://quota-internals`
- `chrome://settings`
- `chrome://stats`
- `chrome://sync-internals`
- `chrome://terms`
- `chrome://tracing`
- `chrome://version`
- `chrome://print`

In the address bar, type `chrome://net-internals`. One use of `net-internals` is to inspect TCP socket events. These TCP sockets are used to transport WebSocket and other protocols used by the browser for communication. When you click **Sockets** on the left, Chrome displays the socket pools. What we're interested in is the currently active, live sockets, so click the **View live sockets** link. In a separate window or tab, open the WebSocket Echo test at <http://www.websocket.org/echo.html>, and click **Connect**. A new entry shows up right away, along with the following URL: <ws://echo.websocket.org/?encoding=text>. Click the entry, and on the right, you'll see the network internals, as shown in Listing A-4.

Listing A-4. Network Internals of a WebSocket Handshake

```
830: SOCKET
ws://echo.websocket.org/?encoding=text
Start Time: 2012-11-23 20:08:27.489

t=1353730107489 [st= 0] +SOCKET_ALIVE [dt=?]
--> source_dependency = 828 (SOCKET_STREAM)
t=1353730107489 [st= 0] +TCP_CONNECT [dt=91]
--> address_list = ["174.129.224.73:80"]
t=1353730107489 [st= 0] TCP_CONNECT_ATTEMPT [dt=91]
--> address = "174.129.224.73:80"
t=1353730107580 [st= 91] -TCP_CONNECT
--> source_address = "10.0.1.5:57878"
t=1353730107582 [st= 93] SOCKET_BYTES_SENT
--> byte_count = 470
t=1353730107677 [st=188] SOCKET_BYTES_RECEIVED
--> byte_count = 542
```


Now, from the window that displays `websocket.org`, let's send a message. The `net-internals` panel refreshes, and shows the number of bytes sent (see Figure A-6).

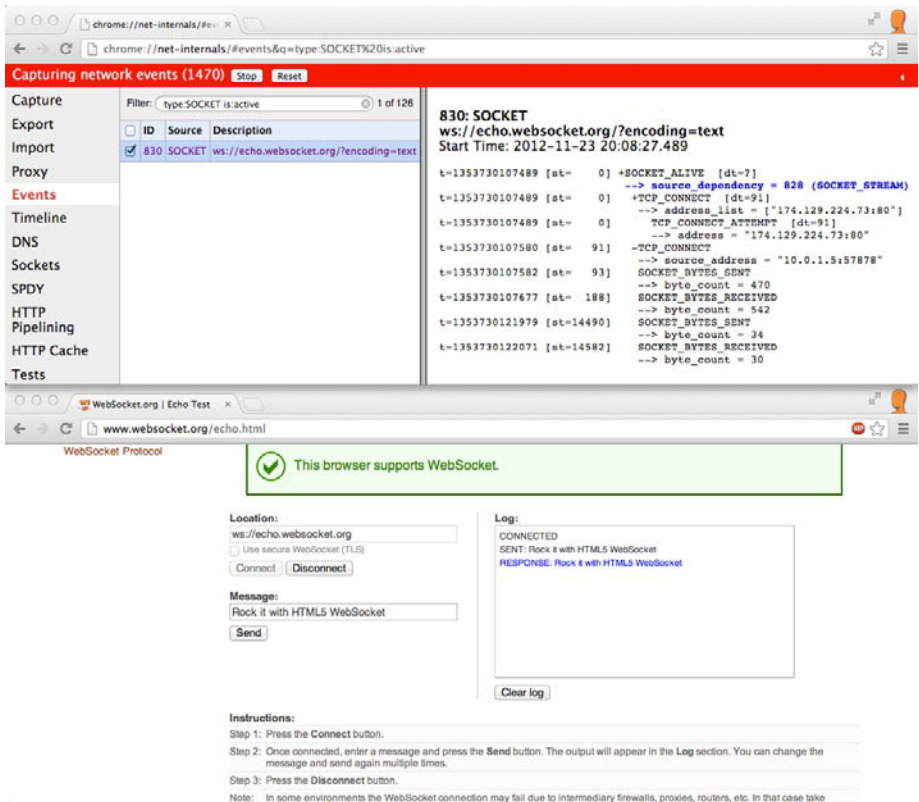


Figure A-6. Google Chrome `net-internals` utility

Much like the Google Developer Tools, `net-internals` is packaged and shipped with Google Chrome. `Net-internals` is a very handy tool if deeper, lower-level network diagnostics are required.

Analyzing Network Packets with Wireshark

Wireshark is a very powerful, free, and open source tool (available for download at <http://www.wireshark.org>) that provides detailed insight into network interfaces, allowing you to see and analyze what's traveling on the wire. Wireshark is a useful tool in WebSocket developers' hands but is widely used by network administrators, as well. Wireshark can capture live network data through the network interface that you can then export/import, filter, color code, and search.

Figure A-7 shows the Wireshark UI as it captures network packets. Under the menu bar and the main toolbar you see the Filter tool bar, which is used to filter the collected

data. This data displays in a tabular format in the packet list pane. The packet details pane shows information about the packet selected in the packet list pane. The packet bytes pane, just above the status bar, displays the packet data, selected in the packet list pane.

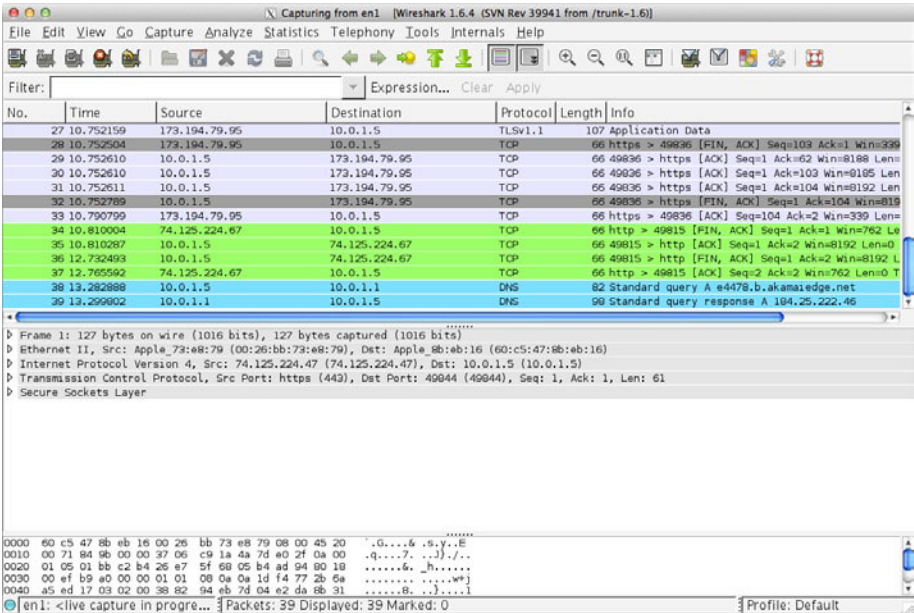


Figure A-7. Wireshark capturing network packets

Start Wireshark and select the network adapter you’re using; if you’re hard-wired to the network, your adapter will be different than when you use WiFi. In our experiment with Wireshark, we’ll inspect the WebSocket traffic between a browser and a WebSocket server, running on [websocket.org](http://www.websocket.org). To get started, navigate with your browser to <http://www.websocket.org>. Then, click the Echo Test link. You can alternatively point your browser directly at <http://www.websocket.org/echo>. Now, you’re ready to establish a WebSocket connection. Click the Connect button.

Since there tends to be quite a bit of traffic on the network, the traffic between your browser and [websocket.org](http://www.websocket.org) quickly scrolls out of view. To ensure we see some useful data, we’ll filter for traffic going to www.websocket.org.

Figure A-8 shows how you can filter out packets with a specific IP address: `ip.dst_host==174.129.224.73`. Wireshark supports the double-equal sign in the condition, as well as the eq operator. In this figure, also notice the WebSocket handshake in the packet details page.

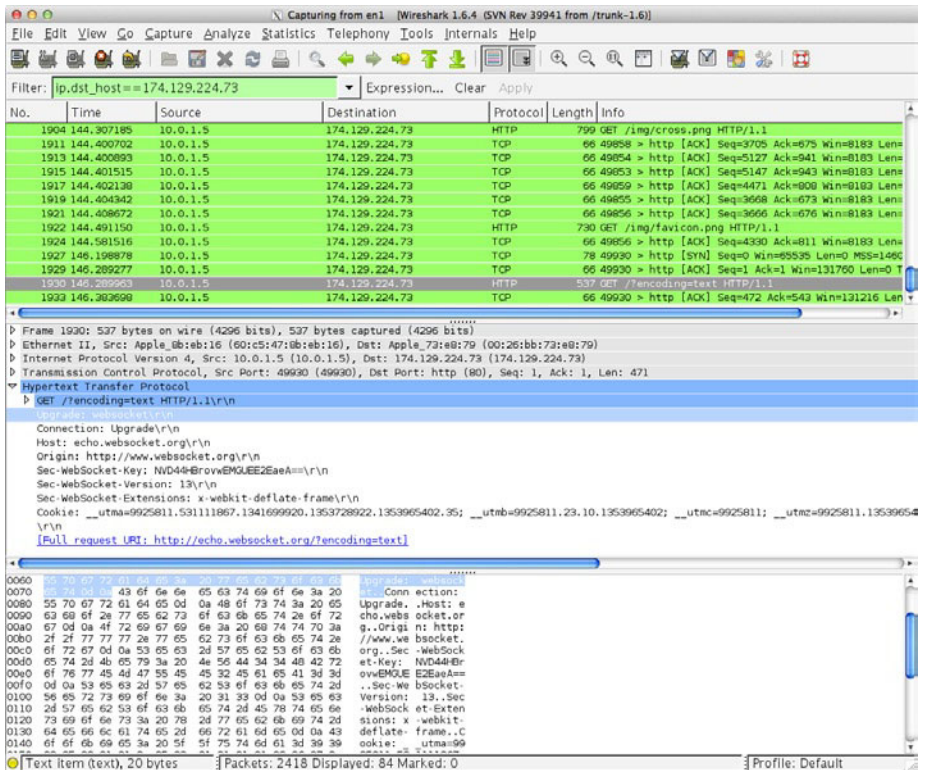


Figure A-8. Filtering network packetsnetwork packets

Another great feature of Wireshark is that it can follow various protocol streams. In Figure A-9 you can see how it follows a TCP stream. It displays the TCP segments that are on the same TCP connection as the selected packet. You can follow a protocol stream by right-mouse clicking on a packet in the packet list pane and choosing Follow from the context menu.

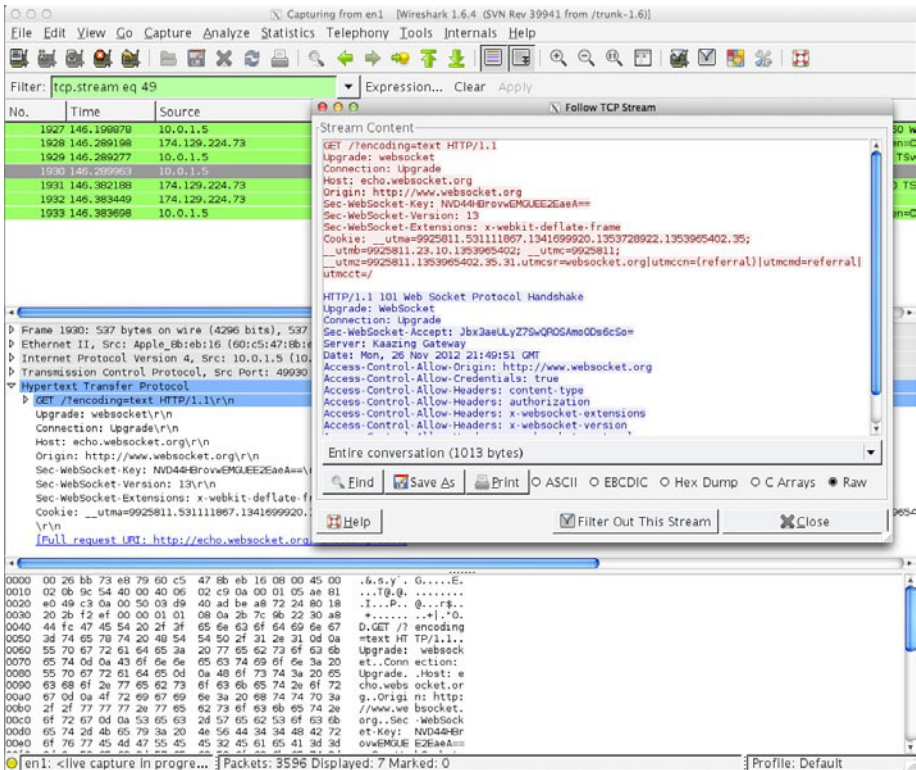


Figure A-9. Following a TCP stream

To see how Wireshark updates the packet list live, submit a WebSocket message in your browser. Figure A-10 shows how submitting the text, *Rock it with WebSocket*, to the Echo service appears in Wireshark.

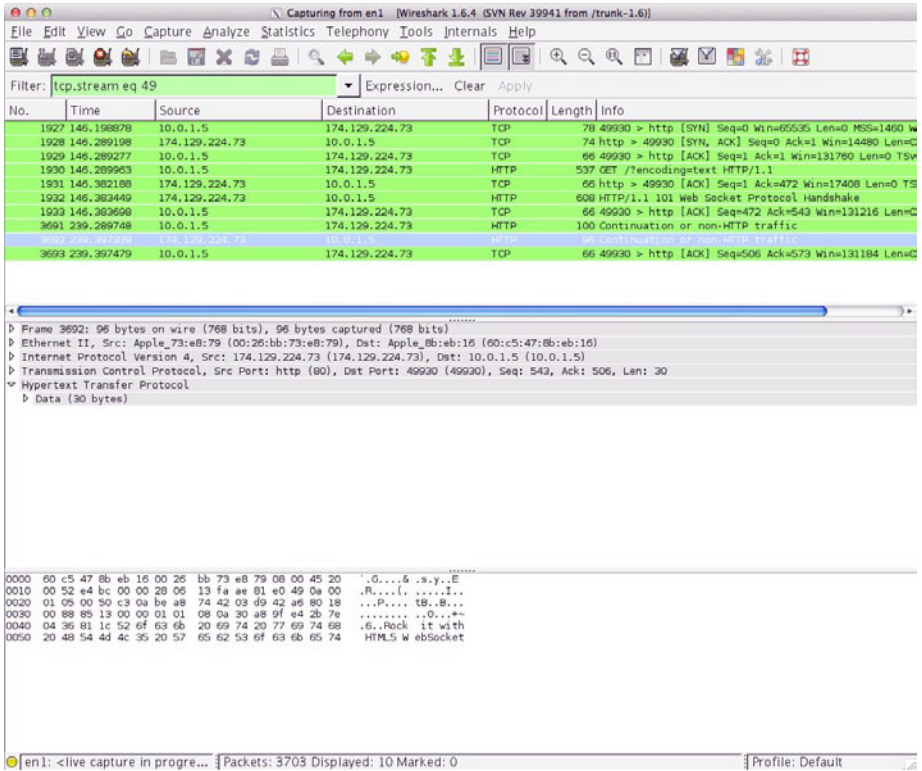


Figure A-10. Wireshark updates live

Summary

In this appendix, we explained some useful tools for inspecting, dissecting, and debugging WebSocket traffic. These tools will help you when building your WebSocket-enabled applications. The next appendix discusses the Virtual Machine (VM) we provide, which includes the open source code (libraries, tools, and servers) we used to build the examples in this book.



WebSocket Resources

Throughout this book, we've used a number of resources that help us build WebSocket applications every day. In this appendix, we walk through how to use the VM (virtual machine) that contains all the code and software pre-installed that you need to build or follow the examples in this book. We also summarize where to get all the libraries, servers, and other technologies we used in this book. Finally, we include a list of WebSocket servers and clients that are available at the time of writing this book.

Using the Virtual Machine

The VM accompanied by this book can be downloaded from the publisher's web site. Simply navigate to <http://apress.com> and search for this book's title (or go directly to www.apress.com/9781430247401). Click the Source Code/Downloads tab and click Download Now. After downloading it, you can start the VM using VirtualBox. VirtualBox is available as a free download from <http://virtualbox.org> for Windows, Mac, Linux, and Solaris host operating systems.

To open the VM, extract it, and double-click the `WebSocketBook.ova` file, or choose **File ► Import Appliance** from the menu of VirtualBox, and select the `WebSocketBook.vbox` file. The operating system of the VM is Ubuntu.

Once you've downloaded and installed the VM, you'll notice a few items on the desktop:

- Icons for Chapters 2-6
- A `README.txt` file

First, open and read the `README.txt`, which explains the servers and services that are automatically started for you when you install the VM. To build the examples described in Chapters 2-6, you can simply start building against the servers and libraries provided in the VM, which are described in the relevant chapter.

Tables B-1 and B-2 describe the servers and libraries that we use throughout the book and whether they are included in the VM.

Table B-1. *Servers Used in this Guide*

Server	Description	Where you can get it	Used in Chapters
Apache ActiveMQ	A popular open source message broker with support for messaging APIs and protocols, like JMS (Java Message Service) and STOMP (Simple or Streaming Text Oriented Messaging Protocol).	http://activemq.apache.org	5 and 7
node-xmpp-bosh	An open source server written by Dhruv Matani that enables XMPP connections over BOSH and WebSocket to any XMPP server. The server is implemented in JavaScript using Node.js.	http://github.com/dhruvbird/node-xmpp-bosh	4
Openfire	An open source RTC (real-time collaboration) server with support for XMPP (Extensible Messaging and Presence Protocol).	http://www.igniterealtime.org/projects/openfire	4
TightVNC	TightVNC is a cross-platform, open source VNC server.	http://tightvnc.com	6
Websocket.org	A publicly hosted WebSocket server with a simple Echo Service for testing and learning about WebSocket.	http://www.websocket.org	1, 3, and 7

Table B-2. *Libraries and Other Tools Used in this Guide*

Library/Tool	Description	Where you can get it	Used in Chapters
jQuery 1.8.2	A widely popular and commonly used open source JavaScript library simplifying cross-browser web development.	http://jquery.com	5
Node.js	A popular open source server for writing applications in JavaScript. Node.js is based on Google Chrome's performant open source V8 JavaScript with support for event-driven asynchronous I/O operations.	http://nodejs.org	3 and 6

(continued)

Table B-2. (continued)

Library/Tool	Description	Where you can get it	Used in Chapters
Node Package Manager (npm)	A Node.js package manager, allowing easy installation of Node.js packages.	http://npmjs.org	None (included in the VM)
Strophe.js	An open source XMPP library for JavaScript, originally created by Jeff Moffitt.	http://strophe.im/strophejs	4
VirtualBox	An open source virtualization product supporting Windows, Mac, Linux, and Solaris as the host operating system, and a significantly larger number of guest operating systems.	http://virtualbox.org	None (used to start the VM)

WebSocket Servers

While you can enable a server to accept WebSocket connections or indeed write your own WebSocket server, there are a few existing implementations that might make your life easier when developing your own WebSocket applications. At the time this book was written, the following are some of the WebSocket servers that are available (list provided by <http://refcardz.dzone.com/refcardz/html5-websocket>):

- Alchemy-Websockets (.NET): <http://alchemywebsockets.net/>
- Apache ActiveMQ: <http://activemq.apache.org/>
- apache-websocket (Apache module): <http://github.com/disconnect/apache-websocket#readme>
- APE Project (C): <http://www.ape-project.org/>
- Autobahn (virtual appliance): <http://autobahn.ws/>
- Caucho Resin (Java): <http://www.caucho.com/>
- Cowboy: <http://github.com/extend/cowboy>
- Cramp (Ruby): <http://cramp.in/>
- Diffusion (Commercial product): <http://www.pushtechology.com/home>
- EM-WebSocket (Ruby): <http://github.com/igrigorik/em-websocket>
- Extendible WebSocket Server (PHP): <http://github.com/wkjagt/Extendible-Web-Socket-Server>

- gevent-websocket (Python): <http://www.gelens.org/code/gevent-websocket/>
- GlassFish (Java): <http://glassfish.java.net/>
- Goliath (Ruby): <http://github.com/postrank-labs/goliath>
- Jetty (Java): <http://jetty.codehaus.org/jetty/>
- jWebSocket (Java): <http://jwebsocket.org/>
- Kaazing WebSocket Gateway (Commercial product): <http://kaazing.com/>
- libwebsockets (C): <http://git.warmcat.com/cgi-bin/cgit/libwebsockets/>
- Misultin (Erlang): <http://github.com/ostinelli/misultin>
- net.websocket (Go): <http://code.google.com/p/go.net/websocket>
- Netty (Java): <http://netty.io/>
- Nugget (.NET): <http://nugget.codeplex.com/>
- phpdaemon (PHP): <http://phpdaemon.net/>
- Pusher (cloud service): <http://pusher.com/>
- pywebsockets (Python): <http://code.google.com/p/pywebsocket/>
- RabbitMQ (Erlang): <http://github.com/videlalvaro/rabbitmq-websockets>
- Socket.io (Node.js): <http://socket.io/>
- SockJS-node (Node): <http://github.com/sockjs/sockjs-node>
- SuperWebSocket (.NET): <http://superwebsocket.codeplex.com/>
- Tomcat (Java): <http://tomcat.apache.org/>
- Tornado (python): <http://www.tornadoweb.org/>
- txWebSocket (Python/Twisted): <http://github.com/rlotun/txWebSocket>
- vert.x (Java): <http://vertx.io/>
- Watersprout (PHP): <http://github.com/chrisnetonline/WaterSpout-Server/blob/master/server.php>
- web-socket-ruby (Ruby): <http://github.com/gimite/web-socket-ruby>

- Webbit (Java): <http://github.com/webbit/webbit>
- WebSocket-Node (Node.js): <http://github.com/Worlize/WebSocket-Node>
- websockify (Python): <http://github.com/kanaka/websockify>
- XSockets (.NET): <http://xsockets.net/>
- Yaws (Erlang): <http://yaws.hyber.org/websockets.yaws>

Index

■ A

Application deployment

abstraction layers

- connection limits, 151
- cross origin deployment, 151
- decreased performance, 151
- emulation and fallback, 150
- full-duplex communication, 150
- messaging, 149
- Modernizr, 151
- non-standard layer, 151
- plugins, 150

buffering and throttling

capacity planning

- monitoring, 159
- planning checklist, 160

distinct classes

encrypted/unencrypted

- WebSocket, 156
- explicit proxy, 156
- firewalls with TLS, 155
- in HTTP and WebSocket, 153
- interaction proxy server, 157
- reverse and load
 - balancing, 152–153
- server as a load balancer, 154
- servers, 155
- server-side intermediary, 152
- SSL termination, 152
- TLS, 158
- transparent proxy, 156
- transport layer security (TLS), 155
- transverse proxy, 155
- web clients, 155
- WebSocket secure (WSS), 157

pings and pongs, full duplex communication, 158

proxies and network

- intermediaries, 152–158
- reverse connectivity, 154
- socket limits, 160
- WebSocket application, 149–151
- WebSocket checklist, 161

■ B, C, D

Business Process Management Systems (BPMS), 108

■ E, F, G

Enterprise Service Bus (ESB), 107

■ H, I

HTML5 WebSocket, 1

applications

- community, 10
- options, 9

connectivity, 2–3

communication

- cross document messaging, 2
- mismatches, 3
- TCPConnection, 3

data traffic

- full duplex communication, 6
- half duplex communication, 6
- HTTP, 3, 101
- HTTP pooling and streaming, 5
- long polling, 6
- polling, 5
- request headers, 4
- response headers, 4
- streaming, 6
- versions, 3

HTML5 WebSocket (*cont.*)

- define, [1-2](#)
- designed to, [1](#)
 - feature areas, [2](#)
 - umbrella term, [2](#)
- high level protocols, [1](#)
- HTTP architecture, [3-6](#)
- initiating
 - connections, [7](#)
 - latency, [7](#)
 - Polling *vs.* WebSocket, [7](#)
- is about HTML5
 - performance, [8](#)
 - simplicity, [8](#)
 - standards, [8](#)
 - web applications, [9](#)
- needs of, [7-9](#)
- real time communication
 - server-sent events, [10](#)
 - SPDY, [11](#)
 - WebRTC, [11](#)
- and RFC 6455, [9](#)
- technologies, [10-11](#)
- world of, [9-10](#)

■ J, K, L

Jabber Instant Messaging (IM), [61](#)

■ M, N

Machine-to-machine (M2M)
communication, [107](#)

■ O, P, Q

Onconnect() callback function, [101](#)

■ R

Remote Framebuffer Protocol, [109](#)
accessing another PC, [110](#)
AJAX applications, [111](#)
Apple Remote Desktop (ARD), [111](#)
computer aided design (CAD), [111](#)
Microsoft's RDP, [111](#)
protocols for access, [111](#)
X11, [111](#)
application enhancement, [127](#)
ArrayBuffers
bindSocketHandlers(), [118](#)

byte streams, [116](#)
canvas element, [120](#)
CompositeStream.append(), [116](#)
CompositeStream.
consume(), [116](#)
connection setting, [117](#)
copyRect() function, [121](#)
doKeyEvent(), [126](#)
DOM keyboard events, [126](#)
efficiency, [122](#)
enable, accept updates, [119](#)
encoding, [122](#)
event, mouse, [123](#)
FrameBuffer requests, [119](#)
hardcoding, [115](#)
HTML, [116](#)
HTML5 application, [113](#)
HTML5 <canvas>, [120](#)
input handling, [122](#)
JavaScript, implement RFB, [116](#)
keyboard input, [125](#)
keydown, [126](#)
KeyEvent(), [125](#)
KeyEvent message, [125](#)
keyup, [126](#)
layers, [113](#)
message types, [122](#)
mousedown, [124](#)
mouse event to VNC server, [124](#)
mouse input, [123](#)
mousemove, [124](#)
mouseover, [124](#)
mouseup, [124](#)
numerical functions, [117](#)
PointerEvent message, [123](#)
protocol library, [113](#)
proxy server, [114](#)
putImageData() function, [120](#)
raw pixel data, [121](#)
RFB client, [115](#)
RfbProtocolClient connect
function, [117](#)
RFB servers, [115](#)
to server messages, [122](#)
transmit a mouse click, [124](#)
versionHandler(), [119](#)
WebSocket.send(), [116](#)
binary and text-oriented
connection over WebSocket, [113](#)
transmit binary image, [112](#)
use over WebSocket, [112](#)

- definition, [111](#)
- description, [111–113](#)
- graphical user interface (GUI), [109](#)
- over WebSocket application, [110](#)
- virtual machine (VM), [110](#)
- virtual network computing (VNC), [110–111](#)
 - client over WebSocket, [113–126](#)
 - code setting, proxy server, [114](#)

■ S, T, U

- Simple Text Oriented Messaging Protocol (STOMP)
 - ActiveMQ, [88](#)
 - Apache ActiveMQ, [91](#)
 - directory structure, [90](#)
 - STOMP over WebSocket, [89](#)
 - URL, [92–93](#)
 - WebSocket Connector, [92](#)
 - Adding interaction, [104–105](#)
 - activemq.xml file, [92](#)
 - game buttons, [102](#)
 - HTML file, [98–99](#)
 - JavaScript code, [100](#)
 - JavaScript library, [97](#)
 - rock-paper-scissors, [97](#)
 - startGame() function, [100–101](#)
 - Web Messaging, [89](#)
 - Apache ActiveMQ, [105–106](#)
 - game buttons, [101](#)
 - JavaScript code, [99](#)
 - onerror callback function, [101–102](#)
 - rock-paper-scissors, [96](#)
 - roshambo, [95](#)
 - sync-up phase, [95](#)
 - BPMS, [108](#)
 - concepts, [107](#)
 - ESB, [107](#)
 - M2M addresses, [107](#)
 - concepts, [93–94](#)
 - extension, [107](#)
 - key players, [85](#)
 - NULL terminated SEND frame, [89](#)
 - publish/subscribe models, [87](#)
 - STOMP/WS application, [96](#), [99–102](#), [104–106](#)
 - Web Messaging, [90–93](#)
 - activemq.xml file, [92](#)
 - publish/subscribe models
 - AMQP, [88](#)

- message distribution techniques, [86](#)
- messaging systems, [88](#)
- topics and queues, [87](#)
- publish/subscribe protocol, [85](#)
- STOMP/WS application, [95](#), [101](#), [105–106](#)
- Web Messaging, [89–90](#), [92](#), [107–108](#)
- WebSocket, [86](#)

SSL, [155](#)

■ V

VNC with RFB protocol, [109](#)

■ W, X, Y, Z

- WebSocket API
 - argument, constructor, [15](#)
 - array as protocol, [16](#)
 - ArrayBuffer message, [18](#)
 - binary message, [21](#)
 - blob messages, [18](#)
 - bufferedAmount attributes, [23](#)
 - call close() method, [22](#)
 - client server connection, [15](#)
 - close-event, [19](#)
 - close() method, [22](#)
 - connection, [14](#)
 - constructor, [14](#)
 - custom protocols, [16](#)
 - error-event, [19](#)
 - events, [17](#)
 - handling and recovery, error, [20](#)
 - message-event, [17](#)
 - message to the server, [20](#)
 - messaging before connection, [20](#)
 - methods, [20](#)
 - object attributes, [22](#)
 - objects, events, [17](#)
 - onclose, [19](#)
 - onerror, [19](#)
 - onmessage event, [18](#)
 - onopen event, [17](#)
 - open-event, [17](#)
 - open event, message, [21](#)
 - open protocols, [16](#)
 - protocol attributes, [24](#)
 - protocol support, constructor, [15](#)
 - readyState attributes, [22–23](#)
 - readyState property, [21](#)
 - registered protocols, [16](#)

- WebSocket API (*cont.*)
 - send() method, 20, 23
 - send updates, 23
 - subprotocols, 15
 - transport layer security, 14
 - wasClean property, 20
 - WebSocket secure scheme, 14
- client application, 28
 - image in client applications, 31
 - Mozilla FireFox, WebSocket display, 30
- client code, JavaScript console, 27
- communicate and message, 25
- connection establish
 - full duplex communication, 13
 - interface, 13
 - messages, 13
 - STOMP, 14
 - XMPP, 14
- fundamentals, 13–14
- gathering, 24–25
- HTML5 media, 28, 30–31
- initialization, 14–24
- support checking, 26–27
- WebSocket protocol
 - AJAX
 - application protocols, 34
 - client connection, 35
 - close code ranges, 49
 - closed codes, 47
 - closing handshake, 46
 - code, frames, 44
 - comet, 36
 - compression extension, 50
 - connection with request, 40
 - decoding text, 45
 - extensions, 50
 - frames, message, 43
 - full duplex communication, 36
 - header, frames, 44
 - HTTP request, 41
 - internet history, 34
 - IRC, 34
 - key response, 42
 - length, 45
 - masking, 46
 - message format, 43
 - multi-frame messages, 46
 - network address translation (NAT), 35
 - Opcodes, 44
 - opening handshake, 40–41
 - PubSubHubbub protocol, 36
 - requests, 49
 - response from server, 41
 - RFC 6455, 43
 - RSS and Atom, 36
 - Sec-Headers, 43
 - server response, 49
 - subprotocols, 49
 - support for others, 49
 - TCP/IP communication, 34
 - TCP socket, 46
 - Telnet, 34
 - text support, 35
 - transport layer security (TLS), 35
 - uniform resource locators (URLs), 34
 - unmasking, payload, 46
 - UTF-8 text encoding, 46
 - web and HTTP, 34
 - XMLHttpRequest, 35
- initialization, 36–40
- byte streams
 - client connection, 37
 - discrete message, 38
 - Google Chrome developer tools, 40
 - internet capabilities, 36
 - open systems interconnection (OSI), 38
 - TCP style network, 36
 - TCP *vs.* HTTP *vs.* WebSocket, 37
 - traffic inspection, 38
 - WebKit, 39
 - WebSocket in Wireshark, 39
- JavaScript with Node.js, 50–51, 56–60
- chat messages
 - code snippets, 51
 - Echo server, 56
 - expression output, 59
 - extensions, 60
 - IO APIs, 50
 - remote JavaScript console, 57–58
 - repl-client.html, 59
 - testing server, 56
 - websocket-repl.js, 58
 - WebSocket server, 51
- protocols, 33–36, 40–50
 - closed codes, 48
 - define, 33

- WebSocket resources
 - Apache ActiveMQ
 - jQuery1.8.2, 178
 - libraries, 178
 - node.js, 178
 - node package manager (NPM), 179
 - node-xmpp-bosh, 178
 - Openfire, 178
 - servers, 178
 - Strophe.js, 179
 - TightVNC, 178
 - tools, 178
 - VirtualBox, 179
 - Websocket.org, 178
 - servers, 179
 - virtual machine, 177–179
- WebSocket security
 - ActiveMQ
 - advanced message queueing protocol (AMQP), 140
 - Apache ActiveMQ message, 140
 - authentication, 141
 - authorization, 144
 - configure, ActiveMQ, 141
 - login with new configuration, 143
 - message broker, 144
 - policy, authorization, 146
 - receive messages, 146
 - sample Apache ActiveMQ, 142
 - send messages, 145
 - setting password, 143
 - STOMP, 140
 - XMPP, 140
 - AJAX
 - cross domain rules, 132
 - Denial of Service, 132
 - DoS, 133
 - handshake, accept keys, 135
 - headers with Sec-prefix, 134
 - HTTP proxies and masking, 136
 - network topology, 137
 - origin header, 131
 - postMessage() API, 132
 - privileged code, 133
 - proxy servers, 136–137
 - RFC 6455, 131
 - scheme, host and port, 132
 - Sec-headers, 135
 - security perspective, 133
 - Sec-WebSocket-Key header, 136
 - server response, 131
 - throttling new connections, 134
 - transparent proxy server, 136
 - unmediated TCP, 134
 - unprivileged code, 133
 - WebSocket Handshake, 131
 - WebSocket traffic, 137
 - XORing, 138
 - application level security, 140–146
 - attacks by API and protocol, 130
 - authentication, 140
 - communication, 129
 - define, 129
 - features, 130–138
 - HTTP
 - unencrypted connection, 138
 - WebSocket secure (WSS), 139
 - WS and WSS, 139
 - STOMP, 129
 - threats, 130
 - TLS, 138–139
 - XMPP, 129
- WebSocket Traffic inspection
 - audits panel
 - creation examining, 165
 - Chrome console,
 - send messages, 168
 - console, 164
 - element panel, 164
 - message using console, 168
 - network panel, 164
 - opening Chrome, 164
 - panel list, 164
 - profile panel, 164
 - resource panel, 164
 - source panel, 164
 - timeline panel, 164
 - WebSocket frames, 167
 - WebSocket handshake, 166
 - capturing packets
 - Echo test link, 172
 - filtering packets, 173
 - follow, TCP stream, 174
 - TCP segments, 173
 - updates, 175
 - with Google Chrome developer tools, 163–168
 - with Google Chrome network internals, 168–171
 - net internals, 171
 - of a WebSocket handshake, 170
 - utilities, 169
 - network packets,
 - Wireshark, 171–175

WebSocket with XMPP

- addHandler, 79
- BOSH, 68
- changes in browser, 75
- chat_app.js, 70, 79
- chat.css, 69
- chat.html, 68
- client library, 68
- client response, 79
- completed application, 79
- connect and start, 68
- connection, 76
- exchange of messages, 75
- handling updates, 74
- iq stanzas, 79
- listen, incoming message, 75
- logging, Pidgin, 71
- message to server, 77
- Pidgin and chat.html, 76
- pings and pongs, 78
- presence and status, 71
- registering handler, 74
- server ping, 78
- stanza, presence, 72
- status message, 72
- status text, 72
- Strophe.js, message with, 77
- test users, 67
- update UI, presence, 72
- web client conversation, 78
- WebSocket enabled server, 67
- XMPP parlance, 72

- chat and instant messaging
 - application, 67–72, 74–79
 - button event, 73
 - not bosh, 68
- connection to XMPP server
 - connectivity strategy, 64
 - federation, 66
 - gateway, 64
 - no stanza to message
 - alignment, 66
 - standardization, 64
 - stanza, 64
 - stanza-to-message
 - alignment, 65
 - subprotocol
 - draft proposal, 66
 - to XMPP server, connection, 65
 - WebSocket handshake, 64
 - world wide web consortium (W3C), 63
 - XmppClient, 65
- extensions
 - Google Talk connection, 82
 - user interface, 82
 - use XMPP, 82
- instant messaging, 61
- layered protocols, 61–63
- internet application layer
 - simple text oriented messaging protocol (STOMP), 62
 - web application layer, 63
- streaming mile of XML, 63–66