The WebSocket Protocol

IETF 80 HyBi WG
Takeshi Yoshino
tyoshino at google dot com
Background

• Evolution of web apps
  – Dynamic and real-time application
  – Webmail, Chat, word processing, etc.

• HTTP is not designed for web apps
  – Large overhead
  – Hanging-GET is necessary for real-time server push
WebSocket is (1)

• New protocol over TCP
  – Opening handshake
    • HTTP-esque request and response
  – Newly defined WebSocket frame

• New API for JavaScript

```javascript
var ws = new WebSocket("ws://example.com/foobbar");
ws.onmessage = function(evt) { /* some code */ }
ws.send("Hello World");
...
```
WebSocket is (2)

- Intended to replace hanging-GET based bidirectional channel
  - Two XMLHttpRequest ➔ One WebSocket

- Full duplex
- Smaller overhead
- Fewer TCP connection
- Simpler API
Other Requirements

• Coexist with HTTP on the same port
  – Use 80/443 which are rarely blocked

• Work with HTTP infrastructure
  – Proxy and firewall

• Allow cross origin connection
  – http://example.com/foo.js establish WebSocket to ws://example.org/chat

• Fit JavaScript programming model
Security Concern

• Cross protocol attack
  – Abuse of WebSocket on browser
    • By malicious JavaScript
    • To attack HTTP server, cache, ...
  – Abuse of XMLHttpRequest
    • To attack WebSocket server

• Port scanning
Protocol Overview

• User-agent establishes TCP
  – Order, reliable transmission, congestion control are guaranteed by TCP
• Opening handshake
• Exchange WebSocket frames
• Closing handshake
Opening Handshake (1) Example

• Client sends

GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBsb2Nvb2utc25jZQ==
Sec-WebSocket-Origin: http://example.com

• Server replies with

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Opening Handshake (2)

• HTTP compliant request/response format
  – Can go through intermediaries for HTTP
  – Code for HTTP can be diverted
• “GET /chat HTTP/1.1”
  – Requested resource is “/chat”
• “Host: server.example.com”
  – Enables name virtual hosting
• “Upgrade” and “Connection” header
  – Tells the server to switch to WebSocket protocol
Opening Handshake (3)
Peer Validation

• Check if the peer is WebSocket ready
  – Only ones understand WebSocket can generate valid Sec-WebSocket-Accept

• Challenge from client : Sec-WebSocket-Key
  – BASE64(Random 16 octets)

• Response from server : Sec-WebSocket-Accept
  – BASE64(SHA-1(concat <Key> and <GUID>))
    – SHA-1 is common, verifiable
    – GUID is uniquely defined for WebSocket
      – “258EAFA5-E914-47DA-95CA-C5AB0DC85B11”
Opening Handshake (4)

• Sec-WebSocket-Origin
  – Optional for non-browser clients
  – Server MAY check
• Sec-* prefix
  – Prevents cross protocol attack with XHR
• Cookie/Set-Cookie as well as HTTP
• Sec-WebSocket-Extensions and Sec-WebSocket-Protocol
  – Discuss later
Framing (1)
Requirements

• Support binary payload
• Single framing for simplicity
  – HyBi 00 used 0x00 <UTF-8> 0xFF for text frame

→ Use payload length field for all type

• Some fields for frame type, extensibility
Framing (2)
Frame Diagram

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>I</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>N</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

opcode | Payload len | Extended payload length |
--------|-------------|--------------------------|
| R       |             |                          |
| S       | (4)        | (16/63)                  |
| V       | (7)        | (if payload len==126/127) |
| 4        |             |                          |

Extended payload length continued, if payload len == 127

Extension data

Application data
Framing (3)
Requirements for Length Field

• Small overhead for small payload
  – Consider power sensitive mobile device
  – Short size like 8 bit is preferred

• Less fragmentation for large data
  – Big range like 64 bit is preferred
Framing (4)
7/16/63 Encoding

• At least 7-bit payload length field
  • 2\textsuperscript{nd} octet of header = RSV4(1), payload\_len(7)
• Extended payload length field may follow

• 0 \leq \text{payload\_len} \leq 125
  – 7 bit
• 126 \leq \text{payload\_len} \leq 2^{16}-1
  – 7 bit + 16 bit extended header
• 2^{16} \leq \text{payload\_len} \leq 2^{63}-1
  – 7 bit + 64 bit extended header
Framing (5)
7/16/63 Encoding

• 63 bit value + 1 bit padding = 64 bit occupation

• Limit is set to $2^{63}-1$ since some platform doesn’t support unsigned 64-bit integer

• Example
  – 5 $\rightarrow$ 0x5
  – 256 $\rightarrow$ 0x7E 0x0100
  – 65536 $\rightarrow$ 0x7F 0x0000000000010000
Framing (6)

Opcodes

• 0x0 Continuation frame
• 0x1 Connection close
• 0x2 Ping
• 0x3 Pong
• 0x4 Text frame
• 0x5 Binary frame
• 0x7-0xFF Reserved
Framing (7)
Room for Extension

• 4 reserved bits in header
  – RSV1, RSV2, RSV3, RSV4
• 9 undefined opcodes 0x7-0xf
• Extension data field
Framing – Open Issue

• Single opcode for control frames or
  Multiple opcodes for each control frames
  – Single control opcode
    1 leading octet of payload is control type
      • Easy to tell intermediaries the frame cannot be fragmented
  – Define the range of control opcodes
  – Multiple opcodes for each control type

• How to specify extension field length
Ping and Pong

• Built-in ping
  – For keep alive, health check, ...

• Alice send ping control

• Bob MUST reply with pong control with the same payload as received ping
Frame Masking (1)
Background

- Security concern raised by Adam Barth

Diagram:

- Malicious script
- Victim browser
- Victim HTTP cache
- Attacker controlled host

Code snippet:

```javascript
<WebSocket opening handshake string>
...some bytes...
GET /foobar.js HTTP/1.1
Host: example.com
...
Malicious data
...
Frame Masking (2)

Background

• Intermediaries designed for HTTP may be poisoned

• Mask client-to-server frame
  – Prevent attacker controlled byte sequence from going over wire
Frame Masking (3)
Current Masking Method

• For each frame
  – Get 4 octets from cryptographically secure random number generator
  – masked_data[i] = clear_text[i] XOR mask[i % 4]
  – send mask and masked_data to server
Frame Masking – Open Issue

• Mask frame or mask payload
  – In-frame masking is less secure?
  – Making whole frame is bad for intermediaries?
• Mask only client-to-server or both direction
  – Debugging is easier if symmetric
• Mask extension field or not
Fragmentation (1)

- Enable sending part of message separately
  - Useful for dynamically generated contents
  - Flush partial data to vacate buffer
- Similar concept as HTTP chunked encoding
- Planned to be used for multiplexing

- Message: complete unit of data on app level
- Frame: network layer unit
Fragmentation (2)

• Use FIN bit and “Continuation” opcode

• Example
  – For message "abcdefg..."
  – Frame1
    • !FIN, opcode=<original opcode>, payload=abc...
  – Frame2
    • !FIN, opcode=CONTINUATION, payload=ijk...
  – Frame3
    • FIN, opcode=CONTINUATION, payload=stu...
Extension (1)

- Negotiate on opening handshake
- Modify payload or even whole frame
- Attach some information
  - as RSV1-4, new opcode or per-frame extension data field
Extension (2)

Negotiation Example

- Applied in order the extensions are listed
- Server accepts part of requested extensions

```
Sec-WebSocket-Extensions: deflate-stream
Sec-WebSocket-Extensions: mux; max-channels=4; flow-control, deflate-stream
Sec-WebSocket-Extensions: x-private-extension
```
Extension – Open Issue

• How to assign reserved bits and opcodes
• How multiple extensions interact
• Intermediaries are allowed to join/split fragmented frames with extension? How?
• Extension may consume unused opcodes?
Subprotocol

- Client may request subprotocol by Sec-WebSocket-Protocol header
- Server choose one from requested subprotocols and echo back it to accept
Closing Handshake (1)

Background

- **WebSocket is full-duplex**
  - Peer may send a frame anytime
- **RST hazard**
  - A peer may close socket without reading out all received data from TCP stack
  - Cause sending RST
  - Peer may miss some data due to RST
- **shutdown(SHUT_WR) is not available everywhere**
- **Implement safe-close on WebSocket layer**
Closing Handshake (2)

- Alice sends close frame to Bob
- Bob sends close frame to Alice
- Bob closes socket
- Alice closes socket

- A peer can close TCP once both received and sent close
Closing Handshake (3)

• What this assures for Alice
  – Alice received all data sent from Bob
    • wasClean parameter of onclose handler
  – It's safe for Alice to close TCP connection
    • No more data coming from Bob ➔ No RST hazard

• What this DOES NOT assure for Alice
  – Bob received all data sent from Alice
    • This requires 3-way close handshake
Status Code

• First two octets of close frame
• Not to be confusing, 4-digit code is used
  – while HTTP uses 3-digit code
  – Predefined codes
    • 1000 Normal closure
    • 1001 Peer is going away
    • 1002 Protocol error
    • 1003 Received unacceptable data
    • 1004 Too large message
• UTF-8 string may follow
Compression
Built-in extension - deflate-stream

- Applies 1951 DEFLATE to whole stream
- Simple
  - No negotiation parameter
  - No reserved bits, opcode, extension data
- Included for now to make sure we have at least one compression available
Compression – Open Issue

• Compress stream, frame or payload
  – Stream compression requires recompression when join/split/insert/filter frames

• More flexibility
  – Per-frame compression parameters
  – More compression algorithms
Gluing with JavaScript

• W3C The WebSocket API
  – http://dev.w3.org/html5/websockets/

• WebSocket class
  – send(), close()
  – onmessage, onclose, onopen, onerror

• To prevent WebSocket from being abused for port scanning, no detail about error occurred during opening handshake will be reported
Gluing with JavaScript – Open Issue

• Specify how to handle error
  – If length field is bad: blah blah
  – If RSV1 is 1: blah blah
  – Pass more information to onerror handler
    • As well as detailed status code now we have

• Interface for binary data handling
  – ArrayBuffer, Blob, ...
  – Ian Hickson is working on this
Other Open Issues

• Keep alive
  – How to maintain underlying TCP connection
    • For long-living WebSocket
    • Have NAT, etc. remember it
    • Ping and pong
  – How to determine ping/pong interval
    • On opening handshake or by some control frame
      – How intermediaries interact
Other Open Issues

• HTTP compliance
  – “Fail on non-101” doesn’t comply HTTP
  – Support redirection
    • Possible security issue
    • Useful for load balancing
  – Reuse, retry of connection after handshake failure

• Multiplexing design