Websockets

Abram Hindle
abram.hindle@ualberta.ca
Department of Computing Science
University of Alberta
http://softwareprocess.es/

Websockets

- Bring two-way communication back to HTTP
- *Do message based communication over HTTP*
- Upgrade existing HTTP connections to a websocket
  - Operate on the same port
  - Use HTTP
  - Try to be HTTP Proxy compatible

WebSocket Examples

- http://www.websocket.org/demos.html
- https://developer.mozilla.org/en-US/demos/detail/
- https://www.youtube.com/watch?v=Ua-PcbC5xMI

Why Websockets?

- You can push
- Observer Pattern is totally possible
- Both sides can send messages back and forth
- Long connections with server-side push
- No need for AJAX polling

RFC 6455

- Websockets
  - Avoid Polling
  - Avoid high overhead messages
    - HTTP Headers are big
  - Reuse existing technologies

Many examples were taken from Fette et al. 2011 https://tools.ietf.org/html/rfc6455

Websocket Handshake

The handshake from the client looks as follows:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

The handshake from the server looks as follows:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

From https://tools.ietf.org/html/rfc6455

Websockets

- Use GET to specify which Websocket
  - 1 webserver can service multiple websocket services
- Connection: Upgrade
  - Not keep-alive
- Update: websocket
  - Let's upgrade to websocket
- Sec-WebSocket-Protocol:
  - How we want to chat?

Finishing that Handshake

- Request:
  - Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
- Response:
  - Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
  - Websocket's GUID:258EAFA5-E914-47DA-95CA-C5AB0DC85B11
  - base64(sha1SumBinary(key+guid))

```
$ echo -en dGhlIHNhbXBsZSBub25jZQ==258EAFA5-E914-47DA-95CA-C5AB0DC85B11 | \
   sha1sum
b37a4f2cc0624f1690f64606cf385945b2bec4ea  -
$ base64 -d | hexdump -C
s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
00000000  b3 7a 4f 2c c0 62 4f 16  90 f6 46 06 cf 38 59 45  |.zO,.bO...F..8YE|
00000010  b2 be c4 ea                                       |....|
00000014
```

Close a Connection

- Send a Close Frame 0x8
  - Upon receiving ignore all new received data.
  - When you receive a close frame, send a close frame and you're done
- Fin/ACK get lost in TCP sometimes.
- Out of band

Ping/Pong

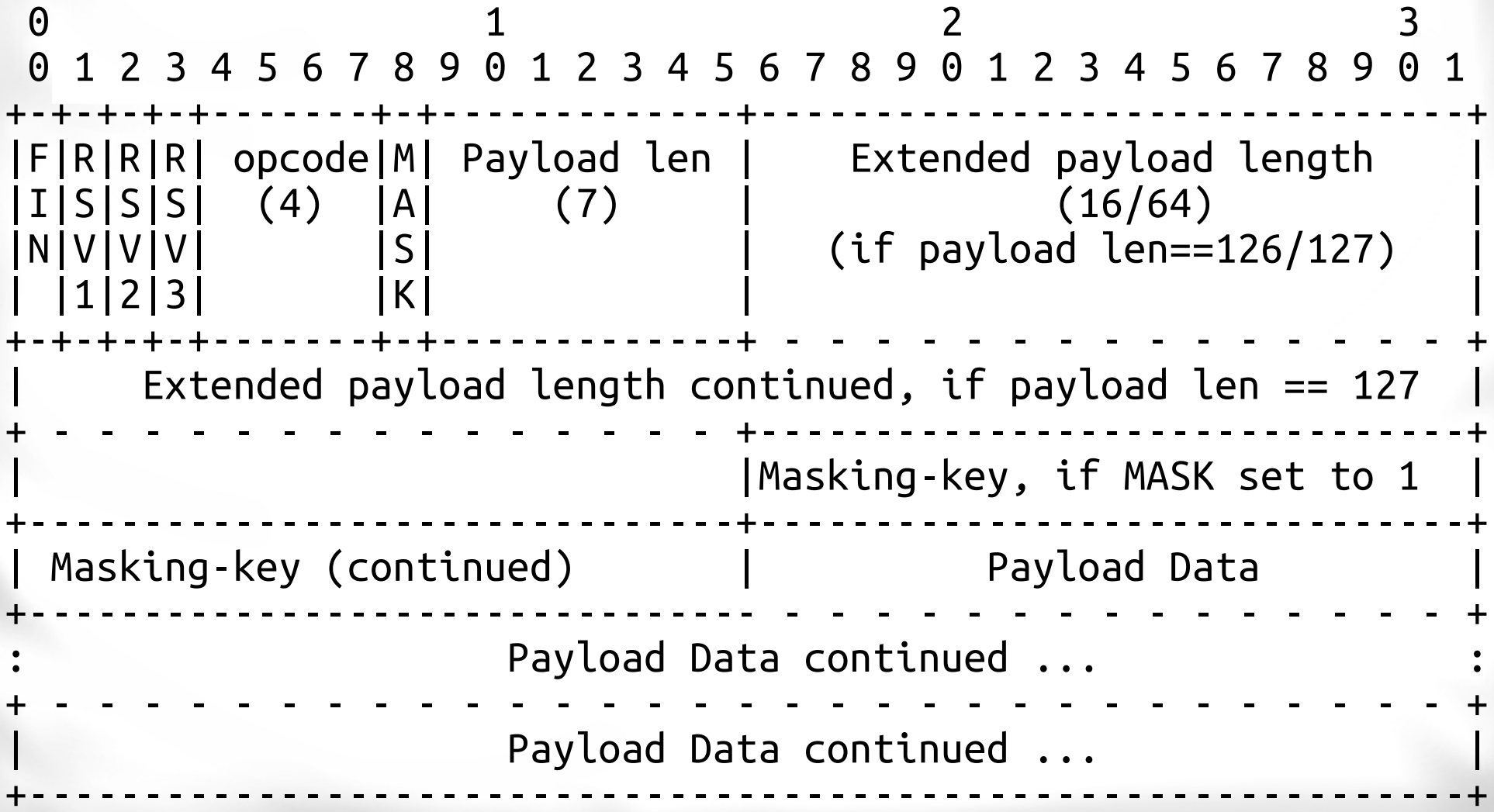
- 0x9 and 0xA control frames
- Used to keep-alive
- Out of Band

Messages versus Dataframe

- Websockets use dataframes to organize and send messages.
- Messages are blobs of text or binary of a particular size. Size is known ahead of time to allow for efficient buffering.
- Messages can be fragmented by dataframes.

Data Frames

- FIN is final fragment in a single message



Many examples were taken from Fette et al. 2011 https://tools.ietf.org/html/rfc6455

DataFrames

- Simplest header is
  - Send Text, no mask size less than < 126
    - 0x81 0b0XXXXXXX
  - Send binary no mask size less than < 126
    - 0x82 0b0XXXXXXX
  - Send big binary no mask
    - 0x82 0x7E 0x0100 – 256 bytes
    - 0x82 0x7E 0x1000 – 4096 bytes
    - 0x82 0x7F 0x0000000000010000 – 65535 bytes
  - Lengths are 7 bits, 16 bits or 64bits
  - Biggest header is about 16+64+32, 112 bits or 14 bytes.

DataFrames

- 0xAB 0xCD 0xEF 0xGH 0xIJ 0xKL 0xMN 0xOP 0xPQ 0xRS 0xTU 0xVW 0xXY 0xZa ~ 112 bits
  - A – Final Packet + 3 reserved bits
    - 0x8 – Final Packet
    - 0x0 – Fragment
  - B – Opcode
    - 0x1 text
    - 0x2 binary
    - 0x8 close
    - 0x9 Ping
    - 0xA Pong
  - 0xCD – Mask + Payload Length
    - 0x8 Mask
    - 0x7E 16bit length
    - 0x7F 64bit length
  - 0xEFGH
    - Could be Extended Payload 16bit
  - 0xEFGHIJKL
    - Could be Mask
  - 0xEFGHIJKLMNOPPQRS
    - Could be 64bit length
  - 0xTUVWXYZa
    - Could be Mask

Data Frames

- Fragments aren't numbered?
  - Why?
  - 0x01 0x03 0x48 0x65 0x6c (contains "Hel")
    - What does the first 0 mean?
    - What odes the first 1 mean
  - 0x80 0x02 0x6c 0x6f (contains "lo")
    - What does that 8 mean
    - Why is it followed by a 0?

Data Frames

- Text (0x1) or Binary (0x2) frames
  - Text is UTF-8 – length in bytes – Do not break UTF-8 characters!
  - Binary is arbitary bytes

Masking

- Websockets are supposed to work with existing infrastructure
- Maintainers worried about cache poisoning by sending fake looking GET requests over websockets. – Bad proxy servers etc.
- Masking encodes and garbles a frame with a mask so that you can't send a GET request in the plain
- XOR the repeated mask against your data
  - Data XOR maskmaskmaskmask....
- Assumption: Client Side is a browser – we want to protect against malicious browser take overs. But internal clients cannot be protected against.

URI!

- You can use ws:yourserver.com:9090/websockethandler/
- wss: is websocket secure (over HTTPS)
- Same format as HTTP URI
  - GET only

Performance

- Better two-way communication
- Missing out on client side caching
- Reinvent the wheel
- Beat the firewall
- Doesn't fully replace XMLHttpRequest

Errors

- Bad UTF-8 encoding → close connection
- No real prescription other than to close connection
- Closing is done by control packet and TLS and TCP close.

Browser as Client

- The browser will not have access to fragments or masking or anything fun like that.
  - Browser API is simple:
    - Open
    - Send and recv messages
    - Close
  - The browser sanitizes everything to make web sockets safe and to avoid exploiting your browser with proxy poisons

WebSocket in JS

- Class: WebSocket
- Constructor:

  new WebSocket("ws://www.example.com/socketserver");

  new WebSocket("ws://www.example.com/socketserver", [“proto1”, “proto2”]);

- Send:

  websocketInstance.send(“A string”);

  var buffer = new ArrayBuffer(16);

  var int32View = new Int32Array(buffer);

  websocketInstance.send(int32View); // send binary

Examples from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays
and https://developer.mozilla.org/en/docs/WebSockets/Writing_WebSocket_client_applications

WebSocket in JS

- Close a connection:

```
websocketInstance.close()
```

- Receive Data:

```
websocketInstance.onmessage = function (event) {
  console.log(event.data);
}
```

Resources

- Mozilla Websocket dev guide
  - https://developer.mozilla.org/en/docs/WebSockets/Writing_
- ByteArrays/Typed Arrays in JS
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/T
- Async Interactions Server Side
  - https://github.com/abramhindle/html5-audio-streaming-via-
- Javascript Example using WebAudio
  - https://github.com/abramhindle/html5-audio-streaming-via-
  -