From WebSockets to WiSH

Wenbo Zhu
Takeshi Yoshino

OSCON 2017
Outline

● The myth behind bidi communication over the Web

● State of confusion
  ○ WebSocket
  ○ HTTP/*

● Pitfalls
Rule #1: avoid bidi communication

1. Causal interaction between the client and server
2. Delivery latency less than network RTT (~10-100ms)
Causal interaction

- In-order message delivery, in both directions
- Request-response semantics still applies

Anti-patterns
1. parallel independent RPCs
   - batching
2. Spontaneous server-push events
   - reverse RPC, or hanging GET or long-polling
Delivery latency < 1 RTT

RTT relative to the speed of light
- L1: ~2x
- L2-3: ~3x
- L4-7: ~10x

Real-time “chat” - 1 Hz
- Sequential RPCs with batching

Online “whiteboard” - 10+ Hz
- High-frequency mutation of shared state
- Games, VR ...
State of confusion

- WebSocket: bidi won’t work with HTTP/1.1
  - Proxies: buffering, idle timeouts ...
  - Web API (XHR)

- HTTP/2: effectively solving a transport (L4) problem
  - Bidi is explicitly allowed (whatwg fetch(streams, grpc ...) ...
  - Server push
WebSockets

RPC handshake + message-framed TCP byte stream

1. TCP/TLS
2. Handshake (RPC)
3. Bidi messages
WebSockets stats

Chrome UMA stats
1. ~7% dynamic HTML pages
2. ~10-100M WS connections
3. distribution of duration
   ○ 50%ile: 20s
   ○ 97%ile: 3600s
4. ~40% connection failures
   ○ ~2.3% over HTTP/HTTPS
   ○ on Windows
Bidi HTTP (full duplex)

Explicit RPC semantics
- Coupled with transport semantics (HTTP/1, HTTP/2, QUIC)
- github.com/bidiweb/http-transport-abstraction

Main Issue
- Buffering: ~1-3% (HTTPS) from experiments
Convergence: Web in Strict HTTP (WiSH)

HTTP/2 as a bidi transport: grpc.io ...

WS-compatible MIME format (bidiweb/wish)
  - application/web-stream

Buffering remains as an issue
  - Proxy awareness
  - Application layer detection
Pitfalls of bidi communication

Short-lived: optimized data delivery
- As safe as stateless RPC

Long-lived => stateful
- Unreliable, data loss due to lack of ack
- Unscalable, lack of load balancing, DoS prevention
Long-lived transactions

Keep-alive
- Timely detection of peer failures is “impossible” over Internet
- Must-have: client-side reconnect, server-side cancellation

Compression
- Per-message compression needs a threshold, e.g. 128B
- Full-body compression is expensive
Long-lived logical transactions

Web prefers shorter transactions (1-10 minutes)

bidiweb/webchannel
- Layer a logical transaction over short-lived transactions
- Load balancing + sticky routing
Fault tolerance

Alive & correct so long as client → cloud is reachable
- Mask transient network/proxy failures
- Mask server endpoint failures
Flow-control

1. Window (credit) based flow-control
2. Transport-level coupling, unique to bidi
   - API survey: bidiweb/bidiweb-semantics

Sender -> protect receivers -> Receiver
Sender -> protect senders -> Receiver
Transport-level (HTTP) proxies
Conclusion

1. Avoid bidi.

2. Websocket is treated as TCP by the Web infrastructure, and HTTP proves to be the best transport for bidi.

3. Long-lived bidi is error-prone by nature.
github/bidiweb

- Best practices, survey
- Portable reference implementations
- API and protocol specs

Q&A

- wenboz@google.com
- tyoshino@google.com