Web-Scale
Bidirectional Communication
(Over HTTP)

Wenbo Zhu (Google)
OSCON 2016
Web-Scale?

Not only about the sheer volume of traffic, but also how a service remains “correct” under ever changing access, load, and failure conditions.
Over HTTP?

Not only about using HTTP/* as a transport, but also how to maintain the key properties of HTTP as a Web protocol

- Request-response (v.s. sockets)
- Resource-oriented (v.s. processes)
- Intermediaries (v.s. point-to-point)
Bi-directional Communication

Client and server exchange messages over a single stateful session.
- Messages are FIFO/programming ordered
- Messages are reliably delivered, or else the session needs be aborted

TCP connection (socket)

1. `channel = open(url)`
2. `channel.send(msg)`
3. `channel.close()`
4. `channel.on("message", callback)`
Shared Whiteboard (goo.gl/zvahKD)
How HTTP Fits?

Whiteboard as shared server state (aka a resource)
- Request-response: causal dependency
- POST: update the state
- GET (polling, push): fetch the latest version of the state
How HTTP Fits? (cont)

Explicit handshake: “OK” to start the communication

Ack/commit semantics: request-response == a single transaction
How HTTP Fits? (cont)

Environments: client platforms, proxies, server-side stacks (frameworks)

Dilemma: messaging pattern (socket like) v.s. communication semantics (HTTP)
A Politically-Correct Implementation

- POST buffered pixels as a single transaction
- GET/POST returns the delta against the local snapshot version
- No interleaved requests
What about the Performance?

- Latency: network RTT (10-1000’s ms)
- Bandwidth: HTTP headers (1-10’s KB)
- CPU: server processing overhead (100 - 10,000’s us)

(demo: 100 msg / sec)
Will HTTP/2 (SPDY, QUIC) Help?

Multiplexed HTTP transactions over a reliable byte-stream transport.

1. Latency: negligible
2. Bandwidth: significant, due to HPACK compression
3. CPU: negligible
Streamed Request-Response Communication

Streamed bodies over a single HTTP transaction.

- Message based delivery: byte-stream => [msg, msg …]
- Full-duplex delivery: explicitly allowed in HTTP/2

→ socket performance + request/response semantics

A single POST Request
What about Web-scale?

Statefulness is always bad: no ack/commit, not reliable, not scalable

Lessons:
1. Evaluate the performance of the stateless model
2. Streaming request/response is good for short & bursting transactions
Survey Conclusion

- Full-duplex/streamed HTTP >= WebSocket
- Web API: whatwg Fetch/streams + Transform functions

WiSH: WebSocket-compatible framing over HTTP body (byte-stream)
WebChannel

A framework to support stateful sessions over HTTP transactions
- API matches WebSocket (JS)
- Leverage HTTP/2: negligible per-request cost
Performance & Reliability

- No long-lived transactions to enable load balancing.
- Real-time failure recovery (reorder, retries, keep-alive)
- Multi-network/path capable
- Multi-homed channels to mask server failures or to support stateless servers
Practical Consideration

1. Flow-control
2. Recovery from peer failures
3. Commit semantics
4. Detect network buffering
5. Randomization & DoS

Bidi-Web Commit Algorithm (#1-3)
Bridge to Cloud Services

- gRPC micro-services: full-duplex streaming, PubSub
- Reverse proxy: Web/Mobile/IoT traffic from Internet
- Kubernetes: automated scaling/deployment/scaling of gRPC services
Q&A

github.com/bidiweb

wenboz@google.com (Wenbo Zhu)

tyoshino@google.com (Takeshi Yoshino)

grpc.io

varuntalwar@google.com (Varun Talwar)