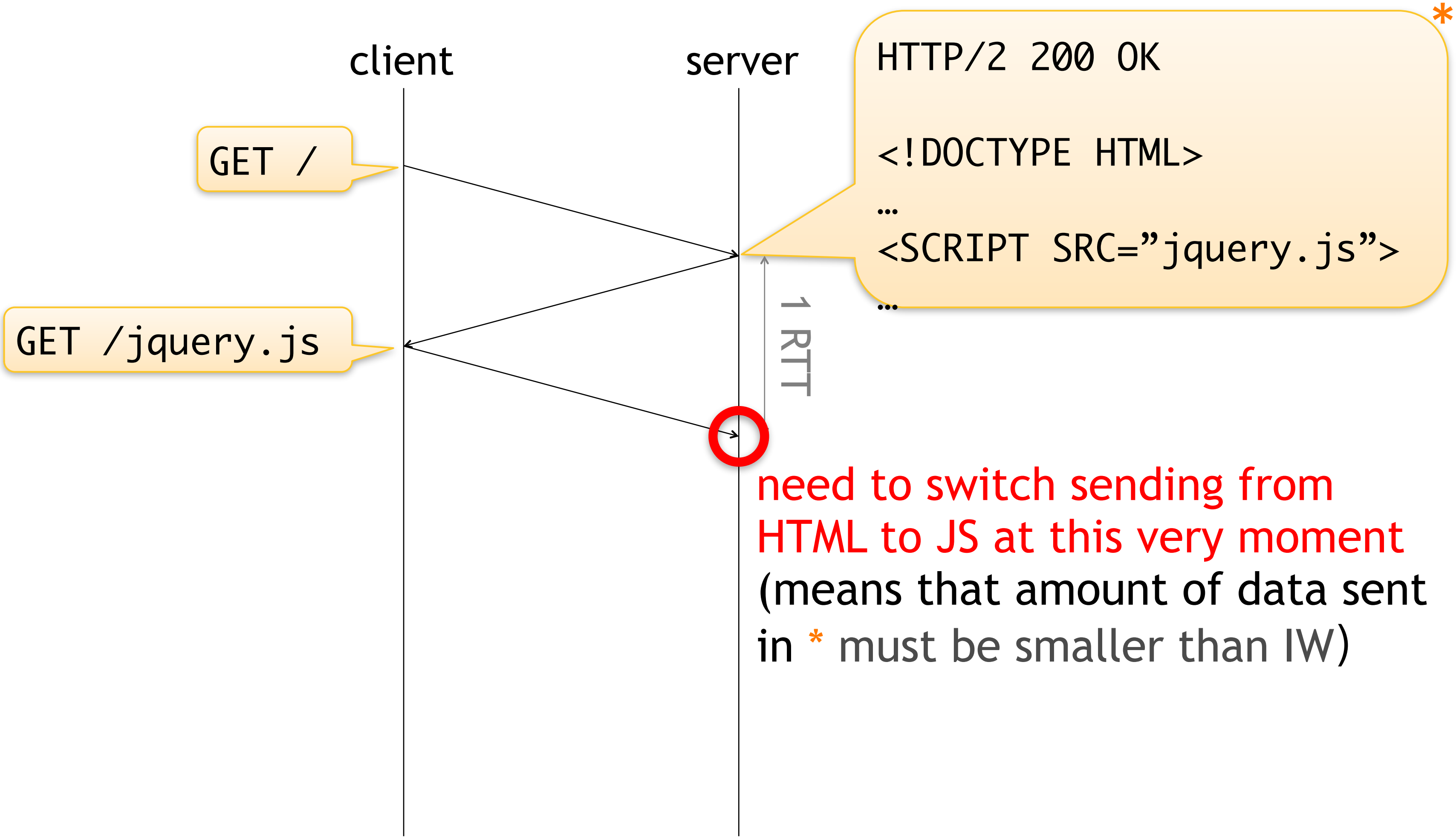
Programming TCP for responsiveness

DeNA Co., Ltd.

Kazuho Oku

Typical sequence of HTTP/2



```
GET /
```

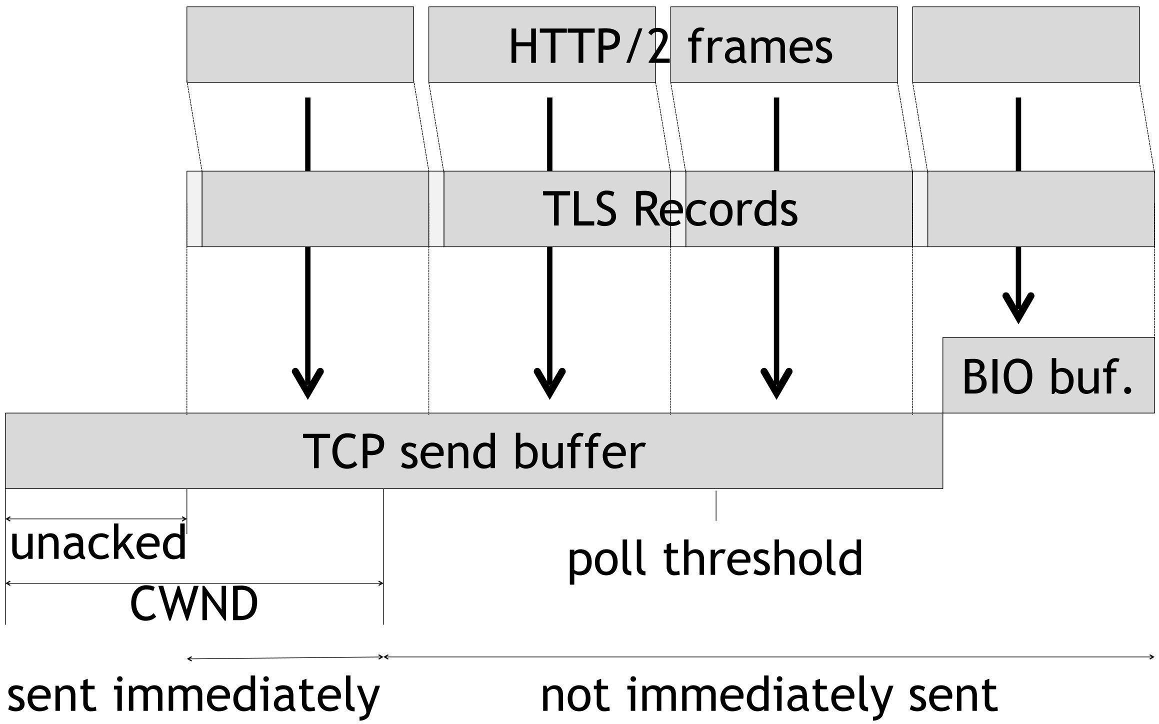
```
HTTP/2 200 OK

<!DOCTYPE HTML>
…
<SCRIPT SRC=”jquery.js”>
…
```

```
GET /jquery.js
```

need to switch sending from HTML to JS at this very moment (means that amount of data sent in * must be smaller than IW)

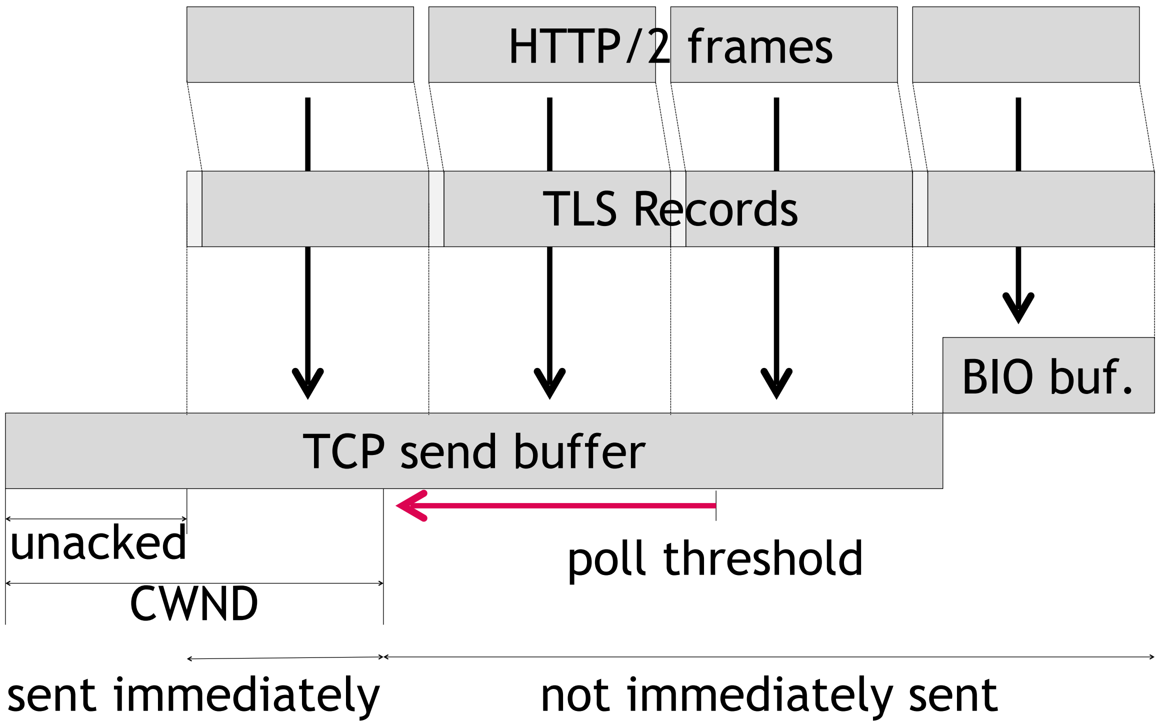
HOB caused by buffers

- TCP send buffer:
  - to reduce ping-pong bet. kernel and application
- BIO buffer:
  - for data that couldn’t be stored in TCP send buffer



Reduce poll threshold

- setsockopt(TCP_NOTSENT_LOWAT)
  - in linux, the minimum is CWND + 1 octet
    - becomes unstable when set to CWND + 0

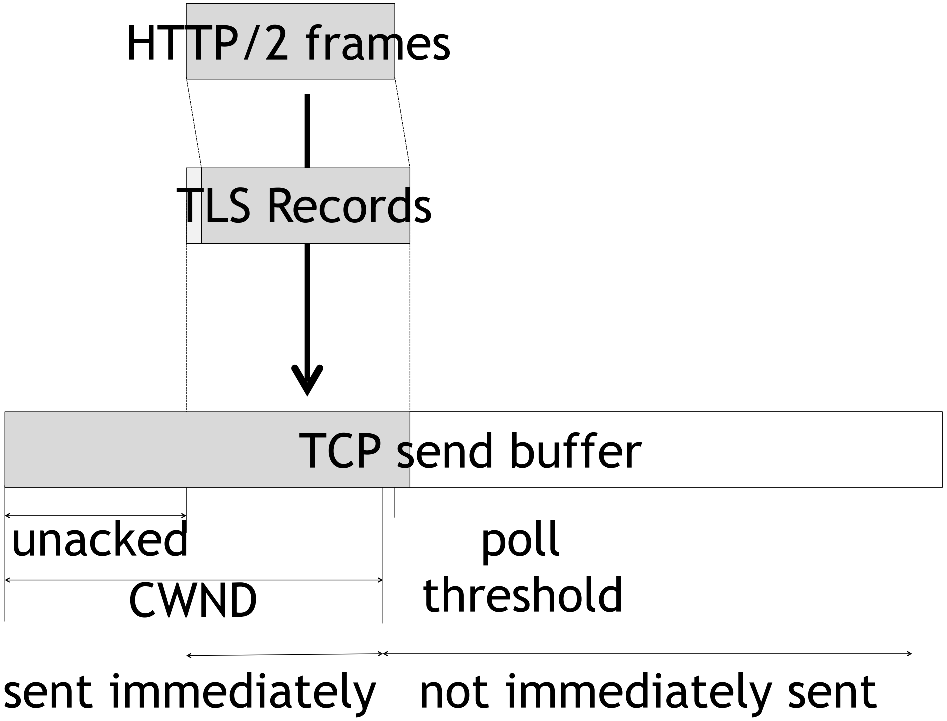


How can we fill up the CWND?

- idea: do smaller writes until epoll tells you its full
- the issue with the idea:
  - CPU intensive
  - data overflowed from CWND might get sent as a small packet (wasting packets during slow start!)
  - overhead of TLS header & HTTP frame becomes bigger

Solution: read TCP states

```
// calc size of data to send by calling getsockopt(TCP_INFO)
if (poll_for_write(fd) == SOCKET_IS_READY) {
    capacity = CWND - unacked + TWO_MSS - TLS_overhead;
    SSL_write(prepare_http2_frames(capacity));
}
```



Negative impact of additional delay

- increased delay bet. ACK recv. → data send, since:
  - traditional approach: completes within kernel
  - this approach: application needs to be notified to generate new data
- outcome:
  - increase of CWND becomes slower
  - leads to slower peak speed?
    - depends on how CWND at peak is calculated
      - does kernel use TCP timestamp for the matter?

Countermeasures

- optimize for responsiveness only when necessary
  - i.e. when RTT is big *and* CWND is small
  - impact of optimization is *RTT * unsent_bytes / CWND*
- disable optimization if additional delay is significant
  - when epoll returns immediately, estimated additional delay is equal to the time spent by the loop

Configuration Directives

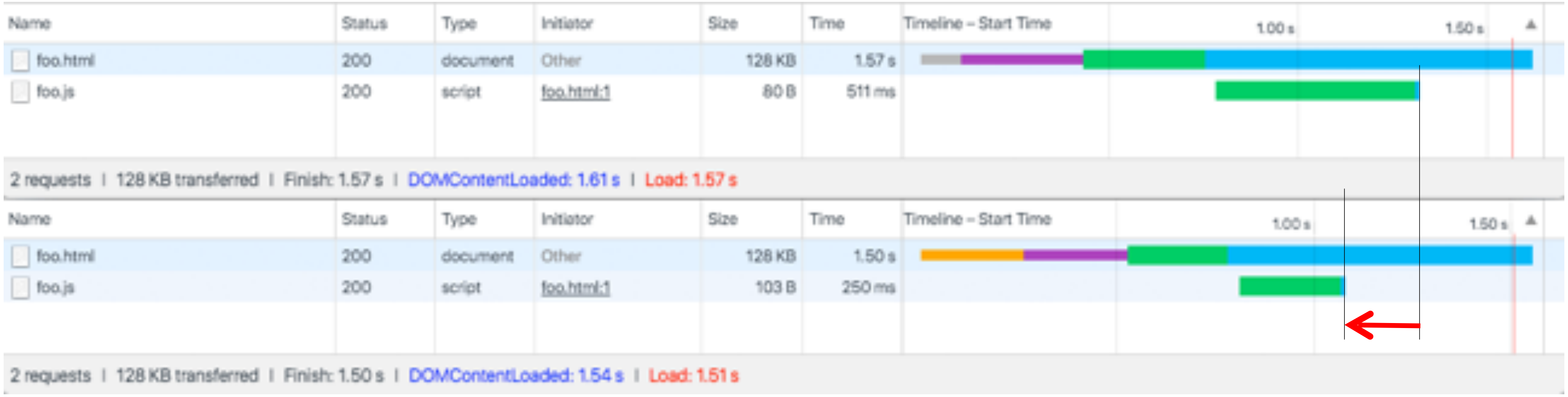
- http2-latopt-min-rtt
  - minimum TCP RTT to enable the optimization
  - default: UINT_MAX (disabled)
- http2-latopt-max-cwnd
  - maximum CWND to enable (in octets)
  - default: 65535
- http2-max-additional-delay
  - max. additional delay (as the ratio to TCP RTT)
  - latopt disabled if the delay is greater
  - default: 0.1

Pseudo-code

```
size_t get_suggested_write_size() {
    getsockopt(fd, IPPROTO_TCP, TCP_INFO, &tcp_info, sizeof(tcp_info));
    if (tcp_info.tcpi_rtt < min_rtt || tcp_info.tcpi_snd_cwnd > max_cwnd)
        return UNKNOWN;

    switch (SSL_get_current_cipher(ssl)->id) {
    case TLS1_CK_RSA_WITH_AES_128_GCM_SHA256:
    case …:
        tls_overhead = 5 + 8 + 16;
        break;
    default:
        return UNKNOWN;
    }

    packets_sendable = tcp_info.tcpi_snd_cwnd > tcp_info.tcpi_unacked ?
        tcp_info.tcpi_snd_cwnd - tcp_info.tcpi_unacked : 0;
    return (packets_sendable + 2) * (tcp_info.tcpi_snd_mss - tls_overhead);
}
```

Benchmark (1)

- conditions:
  - server in Ireland, client in Tokyo (RTT 250ms)
  - load tiny js at the top of a large HTML
- result: delay decreased from 511ms to 250ms
  - i.e. JS fetch latency was 2RTT, became 1 RTT
    - similar results in other environments



Name	Status	Type	Initiator	Size	Time	Timeline – Start Time
foo.html	200	document	Other	128 KB	1.57 s	
foo.js	200	script	foo.html:1	80 B	511 ms	

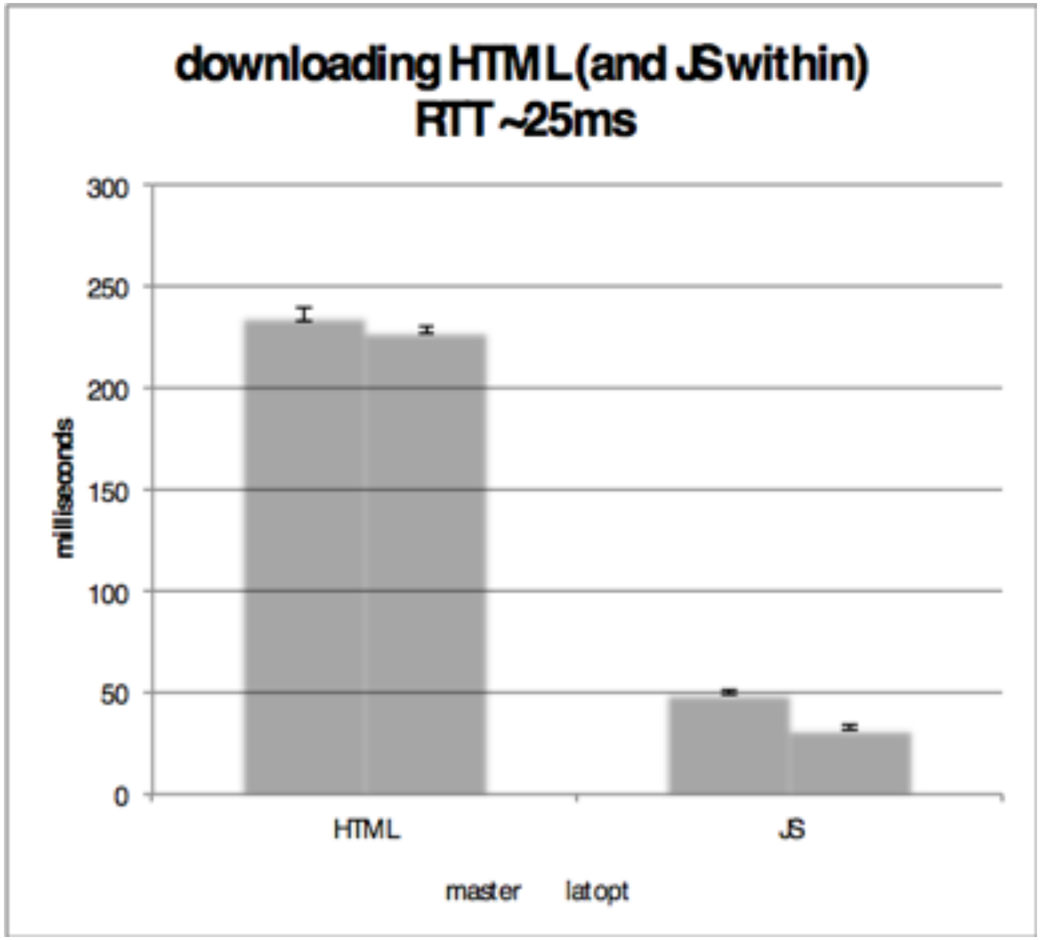
2 requests | 128 KB transferred | Finish: 1.57 s | DOMContentLoaded: 1.61 s | Load: 1.57 s

Name	Status	Type	Initiator	Size	Time	Timeline – Start Time
foo.html	200	document	Other	128 KB	1.50 s	
foo.js	200	script	foo.html:1	103 B	250 ms	

2 requests | 128 KB transferred | Finish: 1.50 s | DOMContentLoaded: 1.54 s | Load: 1.51 s

Benchmark (2)

- using same data as previous
- server: Sakura VPS (Ishikari DC)



Conclusion

- near-optimal result can be achieved
  - by adjusting poll threshold and reading TCP states
  - 1-packet overhead due to restriction in Linux kernel
- 1-RTT improvement in H2O
  - estimated 1-RTT improvement per the depth of the load graph

Under the hood

TCP_NOTSENT_LOWAT

- supported by Linux, OS X
- on Linux:
    - sysctl:
        - set to -1: use kernel default
        - set to 0: sshd hangs
        - set to positive int: override kernel default
    - setsockopt:
        - set to 0: use default (sysctl or kernel)
        - set to int: override default

Unit of CWND

- Linux: # of packets
  - if INITCWND is 10, you can send at most 10 packets at once, regardless of their size
- BSD (incl. OS X): octets
  - you can send CWND*MSS octets, regardless of the number of packets
    - if CWND=10 and MSS=1460, it is possible to send 14,600 packets containing 1-octet payload

Determining amount of data that can be sent immediately

- calculate either of:
  - CWND - inflight
  - min(CWND - (inflight + unsent), 0)
- units used in the calculation must be the same
  - NetBSD: fail

OS	MSS	CWND	inflight	send buffer (inflight + unsent)
Linux	tcpi_snd_mss	tcpi_snd_cwnd*	tcpi_snd_unacked *	ioctl(SIOCOUTQ)
OS X**	tcpi_maxseg	tcpi_snd_cwnd	-	tcpi_snd_sbbytes
FreeBSD	tcpi_snd_mss	tcpi_snd_cwnd	-	ioctl(FIONWRITE)
NetBSD	tcpi_snd_mss	tcpi_snd_cwnd*	-	ioctl(FIONWRITE)

*: units of values marked are packets, unmarked are octets

**: sometimes the values of tcpi_* are returned as zeros