HTTP/2 support for enhanced HTTP client

Communications Subcommittee

Claire Durant

2026 TPF Users Group Conference
March 22-25, Nashville, TN

IBM Z

IBM

HTTP/1: Problem statement

- Calvin's workload: **10,000** HTTP client requests per second to a single server
- Each HTTP/1 connection: **one request at a time**
- **10,000** requests with 100ms response time:
  - **1,000** concurrent requests
  - **1,000** sockets on z/TPF and distributed server
- No big deal for z/TPF
- Painful resource utilization for distributed server

Calvin
capacity planner

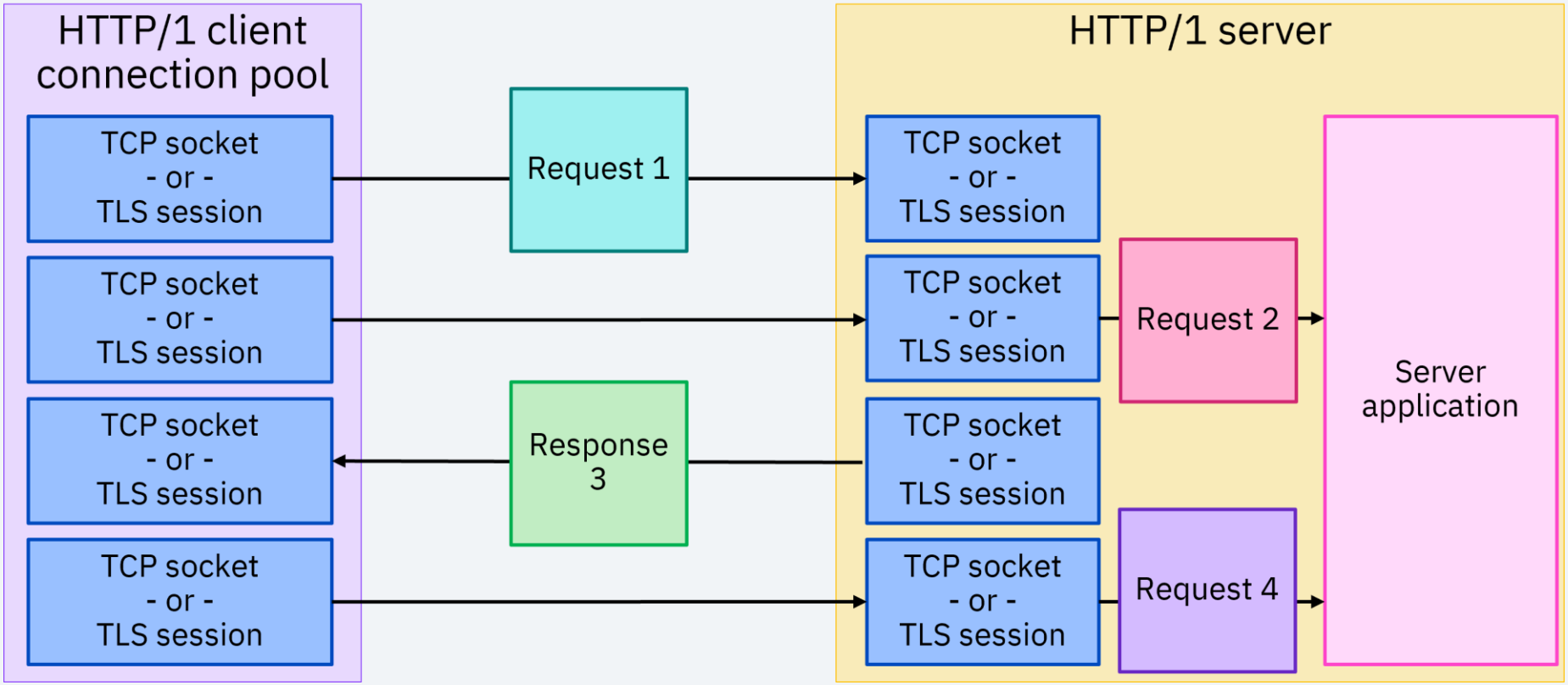
HTTP/2 support for enhanced HTTP client

PJ48066 (March 2026)

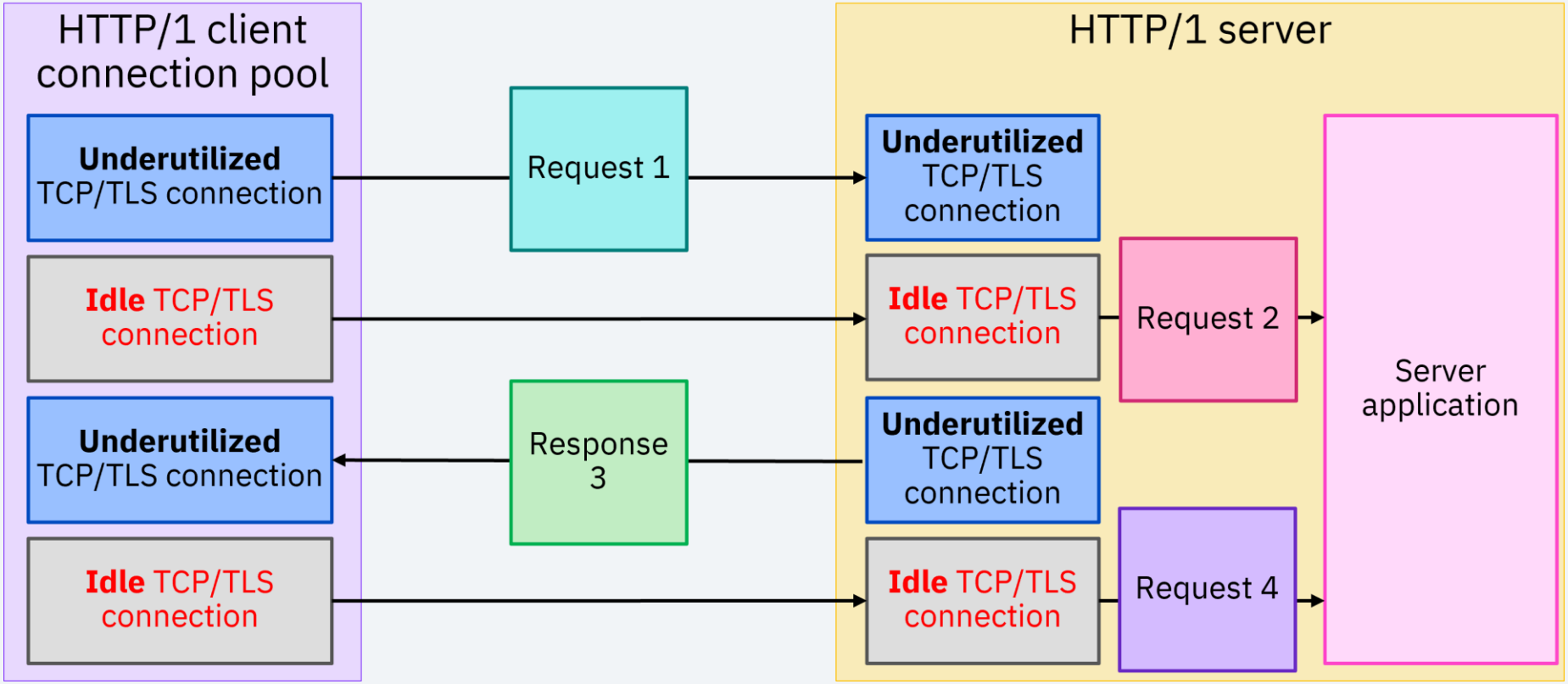
- Adds client-side support for the HTTP/2 protocol on z/TPF
- Fewer sockets required on both z/TPF and remote server
- No application changes required

HTTP/2 overview

Concurrent requests with HTTP/1



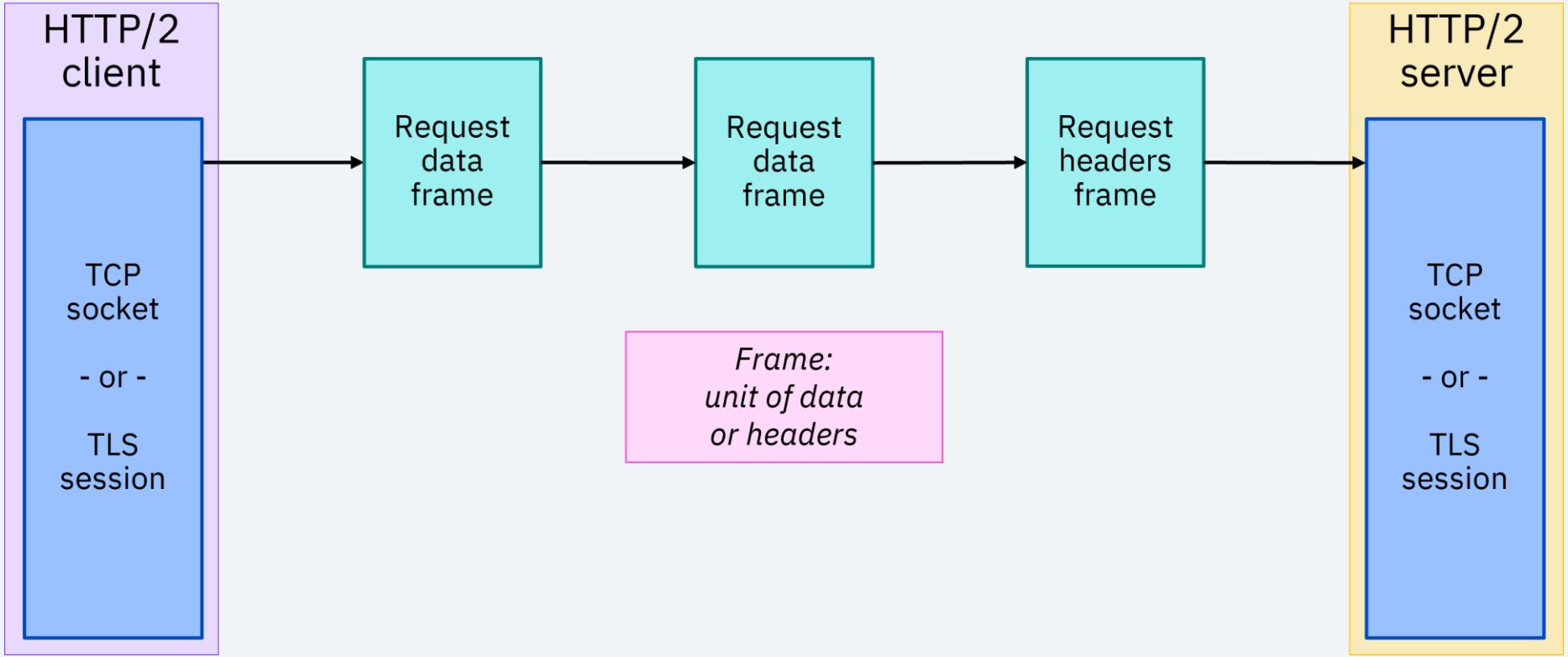
Concurrent requests with HTTP/1



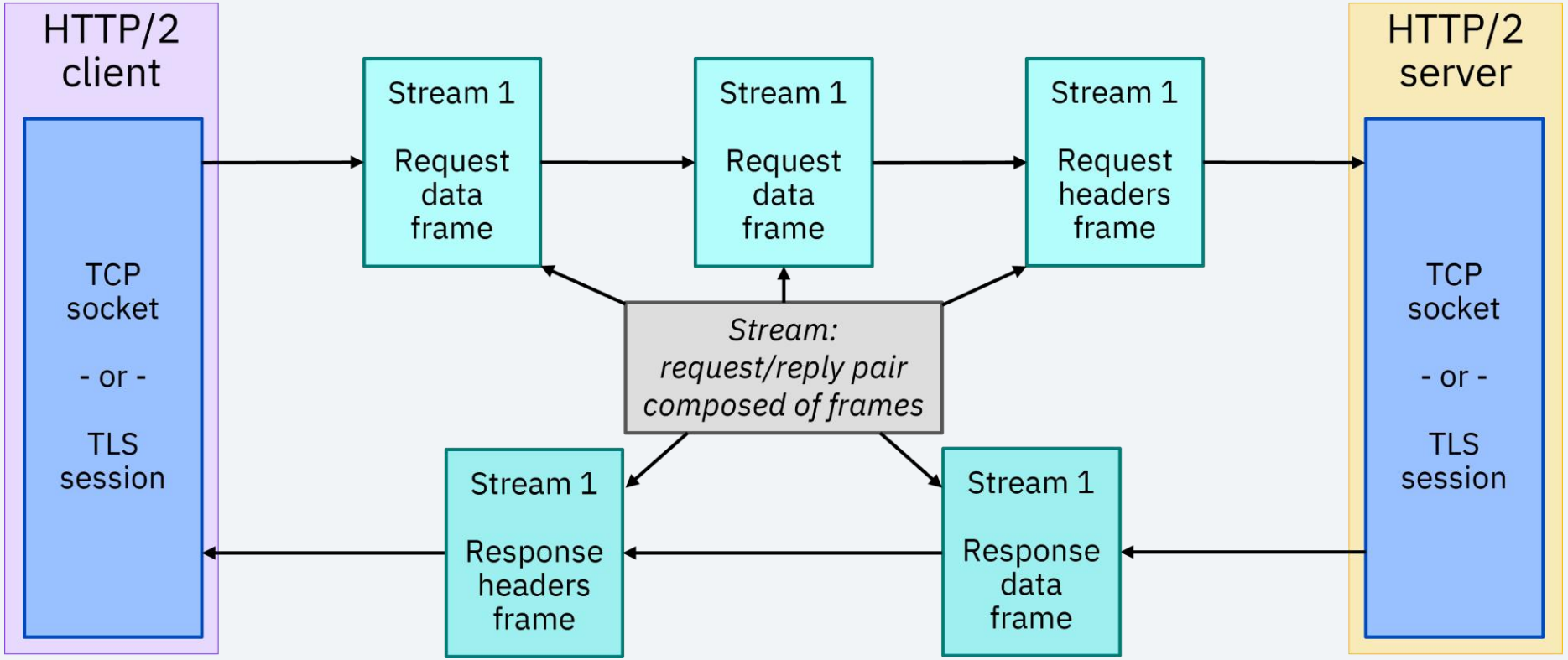
How HTTP/2 works

- Addresses HTTP/1’s inefficient network utilization
  - In HTTP/1, parallel requests require many sockets
- One HTTP/2 connection can carry **multiple** concurrent requests
- Fully **backward compatible** with applications written for HTTP/1

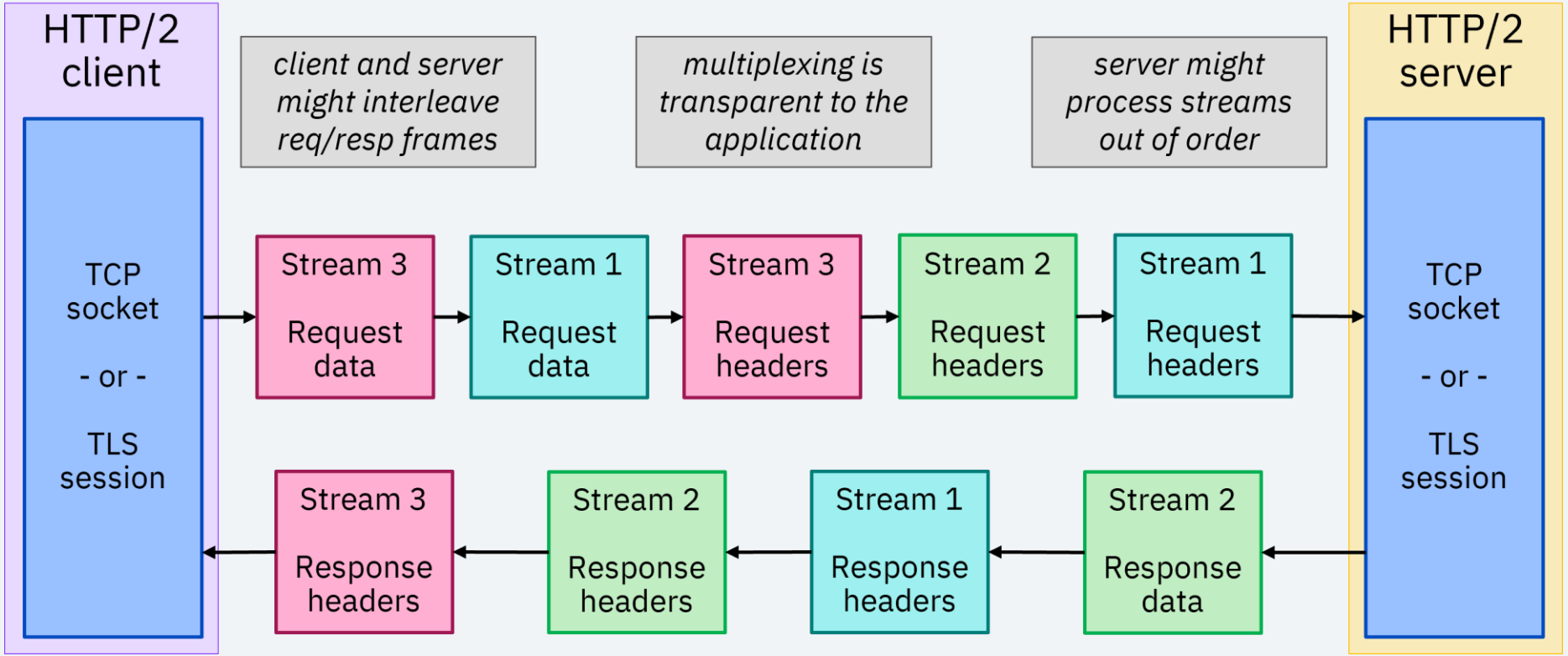
Streams & frames



Streams & frames



Multiplexed streams



Flow control

- HTTP/1: no concurrency → no competing for bandwidth
- HTTP/2: concurrent streams **share** TCP bandwidth
- HTTP/2 introduces flow control on top of TCP’s existing flow control

Flow control windows

- Stream windows: prevent any one stream from dominating connection
- Connection window: prevent combined stream load from overwhelming server
- Automatically handled by z/TPF HTTP/2 client; no tuning required

HTTP/2 server support for z/TPF

- APAR **PJ48022** (March 2025) provided HTTP/2 support for the z/TPF HTTP server
- Link: 2025 TPFUG presentation – z/TPF HTTP server support for HTTP/2

Features & compatibility

Feature parity with HTTP/1 on z/TPF

- Persistent connections with high-speed connector
- TLS support with shared SSL
- Non-persistent sessions
- Synchronous and asynchronous requests
- REST consumer support
- Proxy support
- UHCR logging user exit

Heartbeats

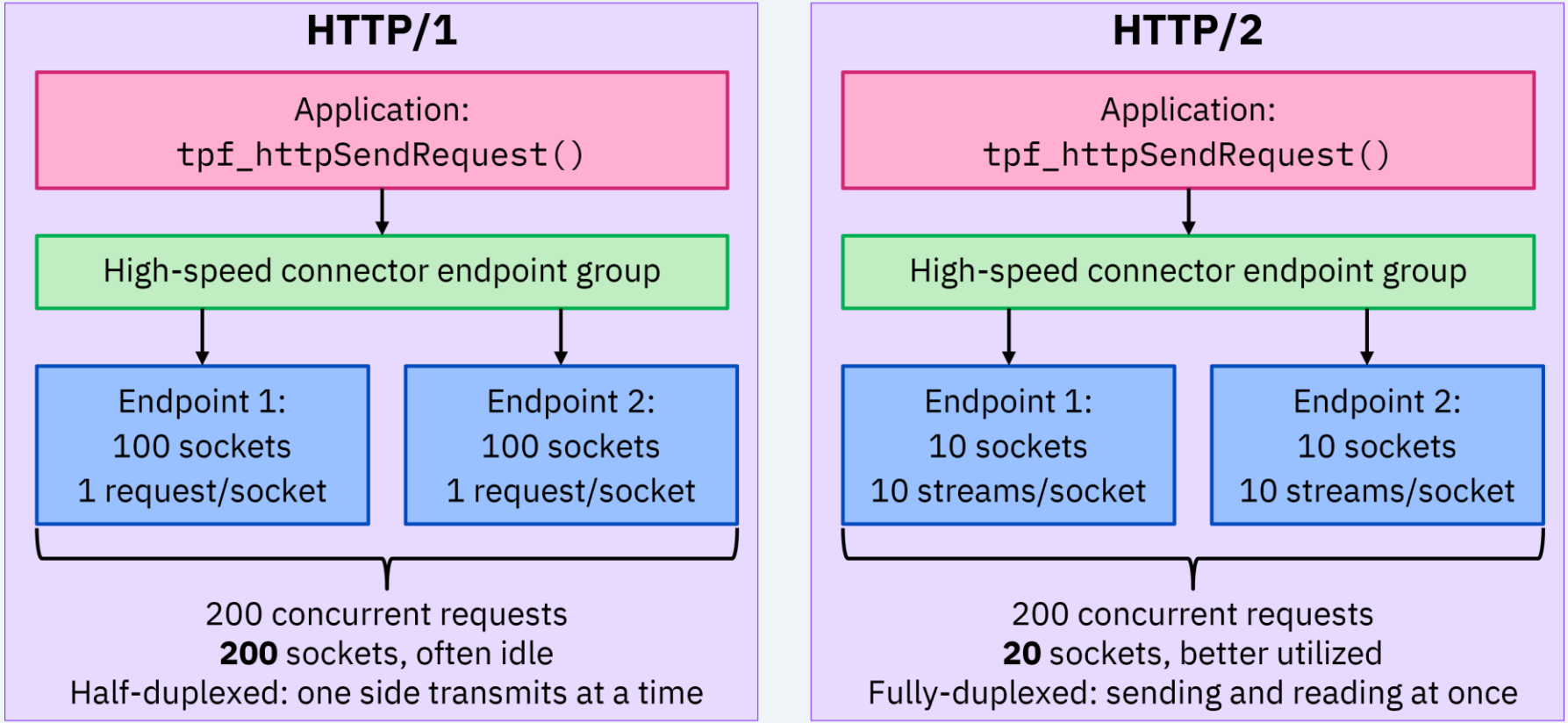
- In HTTP/1, there was no architected way to check health of an idle connection
- HTTP/2 adds protocol-level heartbeat support with PING frames
- Controlled by the `<heartbeatInterval>` endpoint group descriptor attribute
- PING frames are sent only if connection is **idle** for the configured heartbeat interval

Enabling HTTP/2 for persistent sessions

- **No application changes**
- Each high-speed connector endpoint group supports either HTTP/1 **or** HTTP/2
- For easiest migration, server should support both HTTP/1 **and** HTTP/2
- Set endpoint group descriptor `<groupType>` to `HTTP2`
- To migrate existing group: `ZCONN STOP/START`, or `IPL`

Load balancing

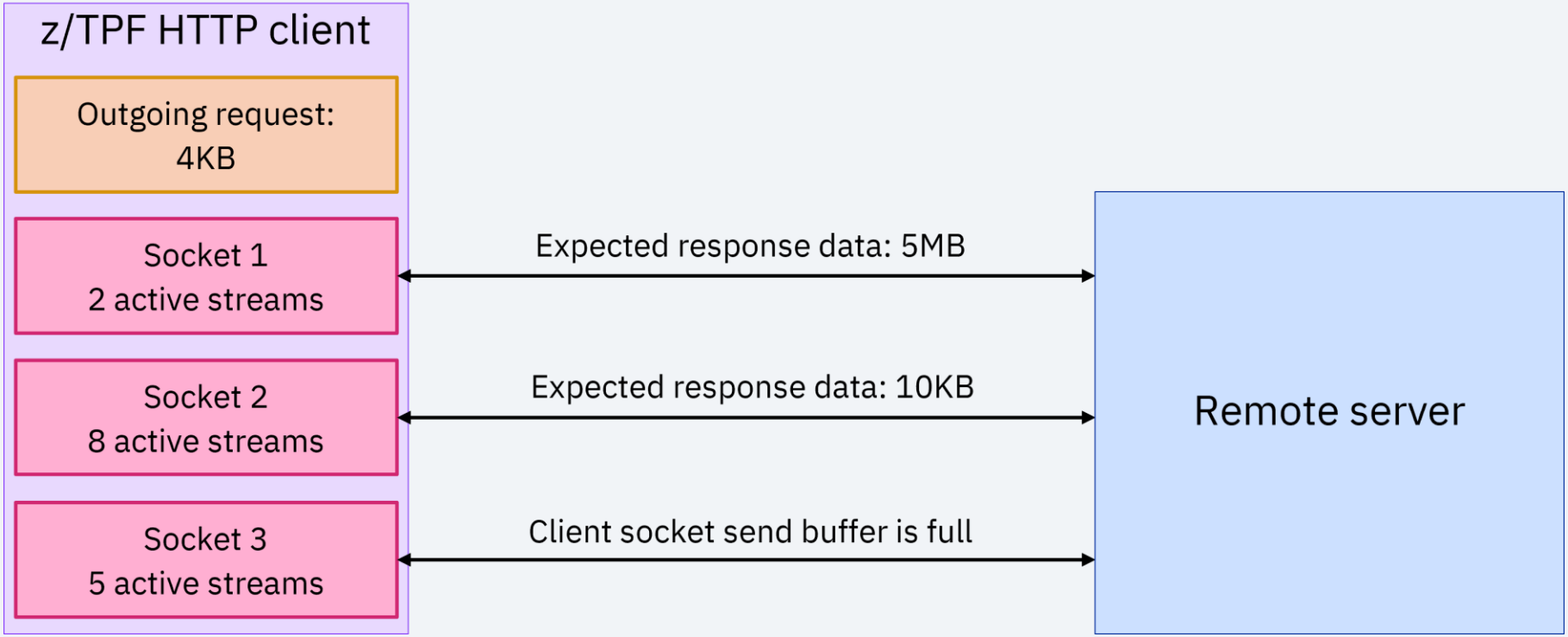
Socket utilization: high-speed connector



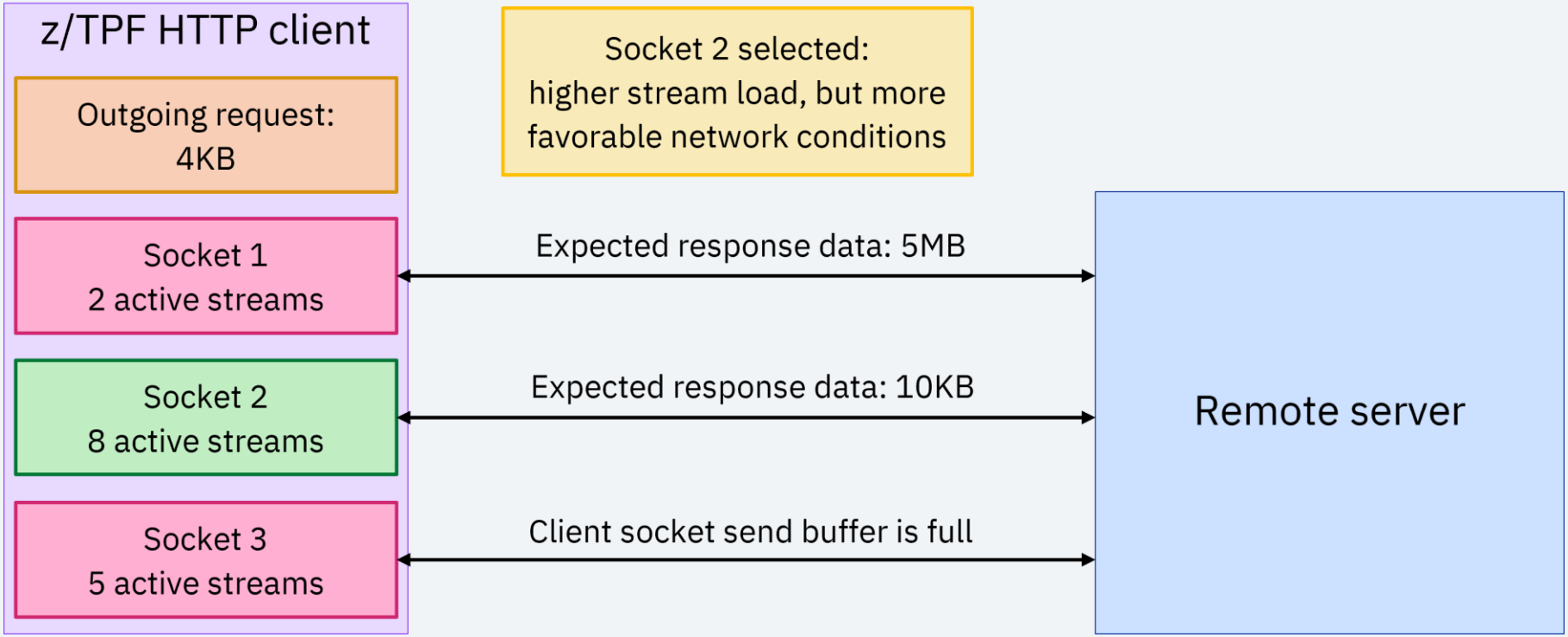
Socket selection

- HTTP/1 uses simple FIFO/LIFO socket selection
- HTTP/2 requires a more intelligent selection algorithm
- Considers stream load, real-time network conditions, and expected data volume
- Optimizes for overall latency and throughput, not perfect balancing of streams
- Endpoint selection is still round-robin

Example: Socket selection



Example: Socket selection



Socket expansion

- HTTP/1: simple expansion – *open a new socket if all in-use*
- HTTP/2 adds **proactive expansion**
  - Open new connections when stream load nears limits, or congestion detected
- HTTP/2 supports similar **reactive expansion** as HTTP/1, in case of a sudden flood of traffic

Observability & tuning

ZCONN statistical displays

- All existing ZCONN statistical commands work with HTTP/2
  - ZCONN STATS
  - ZCONN MAXSTATS
  - ZCONN CLEARSTATS
  - ZCONN DISPLAY
  - ZCONN TOPOLOGY

ZCONN STREAM

SERVER ENDPOINT	SOCKET	MAX CLIENT STREAMS	MAX SERVER STREAMS	ACTIVE STREAMS	HIGH STREAMS	HIGH STREAM TIME
endpt1	00C00332	10	100	5	7	2026/01/23 22:52
endpt1	00C00334	10	100	7	8	2026/01/23 23:10
endpt1	00C00335	10	100	8	8	2026/01/23 23:13
endpt1	00C00338	10	100	7	8	2026/01/23 23:01
endpt1	00C0033A	10	100	6	7	2026/01/23 22:14
endpt2	00C0033C	10	50	7	7	2026/01/23 23:11
endpt2	00C0033E	10	50	8	8	2026/01/23 23:21
endpt2	00C0034D	10	50	7	8	2026/01/23 22:36
endpt2	00C0034F	10	50	7	7	2026/01/23 22:43
endpt2	00C00351	10	50	8	8	2026/01/23 23:09

Effective stream limit:
lesser of client and server maximums

ZCONN FLOW

SERVER ENDPOINT	SOCKET	NUMBER TCPBLOCKS	TIME TCPBLOCKED	NUMBER HTTPBLOCKS	TIME HTTPBLOCKED
endpt1	00C003C4	0	N/A	0	N/A
endpt1	00C003D8	0	N/A	0	N/A
endpt1	00C003DA	0	N/A	0	N/A
endpt1	00C003DB	0	N/A	0	N/A
endpt1	00C003DC	0	N/A	0	N/A
endpt2	00C003E2	2	2026/01/23 23:11	0	N/A
endpt2	00C003E4	4	2026/01/23 23:06	0	N/A
endpt2	00C003EA	3	2026/01/23 23:10	0	N/A
endpt2	00C003EC	1	2026/01/23 23:19	0	N/A
endpt2	00C003EE	5	2026/01/23 23:16	0	N/A

ZCONN FLOW

SERVER ENDPOINT	SOCKET	NUMBER TCPBLOCKS	TIME TCPBLOCKED	NUMBER HTTPBLOCKS	TIME HTTPBLOCKED
endpt1	00C003C4	0	N/A	0	N/A
endpt1	00C003D8	0	N/A	0	N/A
endpt1	00C003DA	0	N/A	0	N/A
endpt1	00C003DB	0	N/A	0	N/A
endpt1	00C003DC	0	N/A	0	N/A
endpt2	00C003E2	2	2026/01/23 23:11	0	N/A
endpt2	00C003E4	4	2026/01/23 23:06	0	N/A
endpt2	00C003EA	3	2026/01/23 23:10	0	N/A
endpt2	00C003EC	1	2026/01/23 23:19	0	N/A
endpt2	00C003EE	5	2026/01/23 23:16	0	N/A

Why is endpt2 getting blocked on the TCP layer?

- streams overloading connection; contention over bandwidth
- request data exceeds send buffer size
- unhealthy connection; server reading data too slowly

Tuning parameters

- `<maxStreams>` → maximum concurrent active streams
- Consider **reducing** the following to limit active sockets:
  - `<startSocket>` → initial number of connections
  - `<maxSocket>` → maximum number of connections
- Consider **increasing** the following to handle bandwidth usage of many active streams
  - `<bufferSendSize>` → TCP send buffer size
  - `<bufferReceiveSize>` → TCP receive buffer size

How many sockets and streams do I need?

- **It depends**
- Conservative approach:
  - `<maxStreams>` → 10 streams per socket
  - `<maxSocket>` → keep as-is
  - `<startSocket>` → reduce by 10x
- Adjust as appropriate based on latency, flow control indicators, stream distribution, etc.

Non-persistent connections

Non-persistent HTTP/2 sessions

- Enhanced HTTP client APIs support non-persistent connections to a remote host
- Connection is established to handle just one request
- Specify HTTP_2 on enhanced HTTP client APIs to use a non-persistent HTTP/2 connection
- Each non-persistent HTTP/2 connections supports only **one** active stream; no multiplexing

Conclusion

Calvin’s story with HTTP/2

- Calvin’s workload: **10,000** HTTP client requests per second to a single server
- Each HTTP/2 connection: ~**10 concurrent streams**
- **10,000** requests with 100ms response time:
  - **1,000** concurrent requests
  - **100** sockets on z/TPF and distributed server
- Resource utilization associated with open sockets is **significantly reduced** on distributed server

Calvin
capacity planner

Key takeaways

- HTTP/2 client support with full HTTP/1 feature parity
- Fewer sockets required on z/TPF and remote servers to serve the same high volume of traffic
- Efficient connection management for low latency with mixed request and response sizes
- Simple migration; no application changes required

Thank you