z/TPF HTTP Server Support for HTTP/2 (PJ48022)

Web Services Subcommittee

Bradd Kadlecik

2025 TPF Users Group Conference
May 5-7, Austin, TX

IBM Z

IBM

Executive Summary

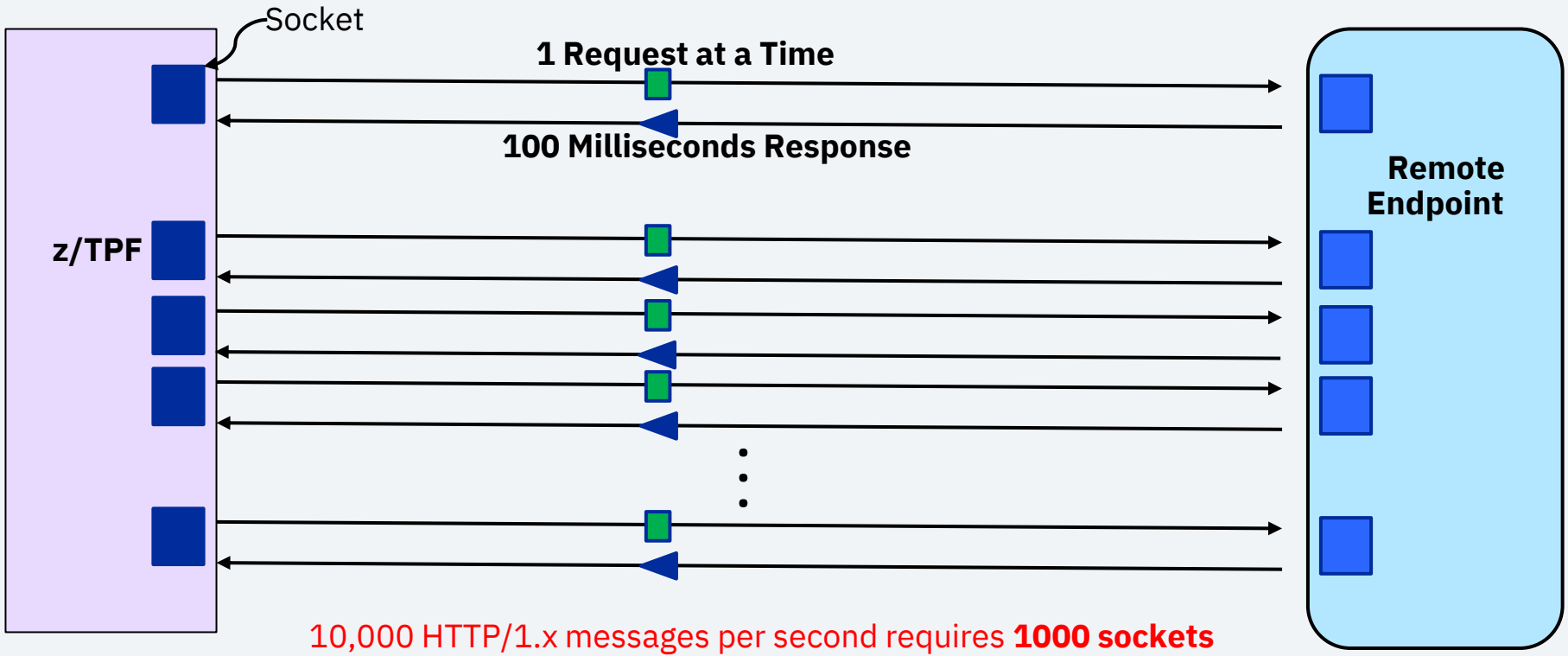
Yumi
Chief Technology Officer

This support reduces the number of sockets required when a large number of HTTP messages are exchanged between a remote client and the z/TPF system. No application changes are required.

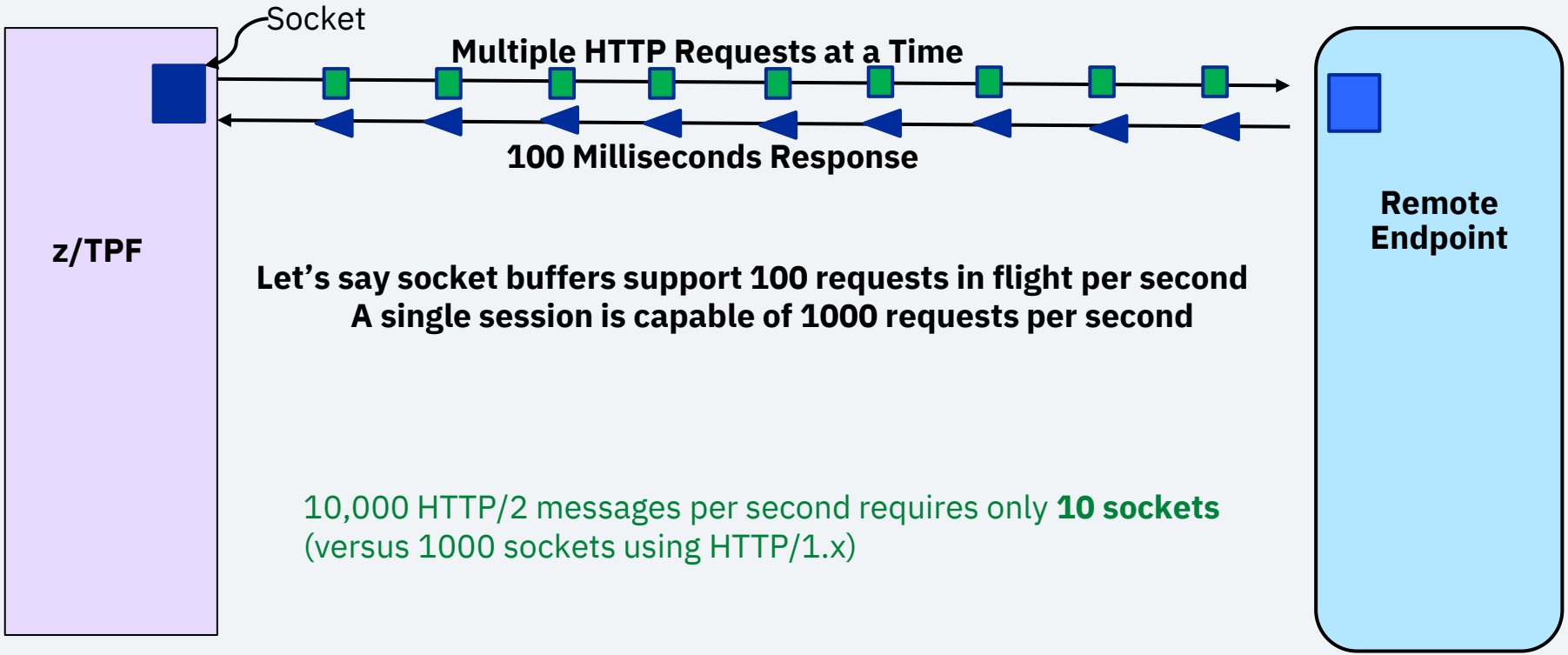
Problem Statement

When using REST with the HTTP/1.x single request/reply model, a large number of sockets must be maintained for a large number of requests between a remote client and the z/TPF system.

As-Is


Socket
1 Request at a Time
100 Milliseconds Response
z/TPF
Remote Endpoint
10,000 HTTP/1.x messages per second requires **1000 sockets**

To-Be



Let's say socket buffers support 100 requests in flight per second
A single session is capable of 1000 requests per second

10,000 HTTP/2 messages per second requires only **10 sockets**
(versus 1000 sockets using HTTP/1.x)

Value Statement

HTTP/2 reduces the number of sockets required when a large number of HTTP messages are exchanged between a remote client and the z/TPF system. No application changes are required.

Technical Details
HTTP/1.x v HTTP/2

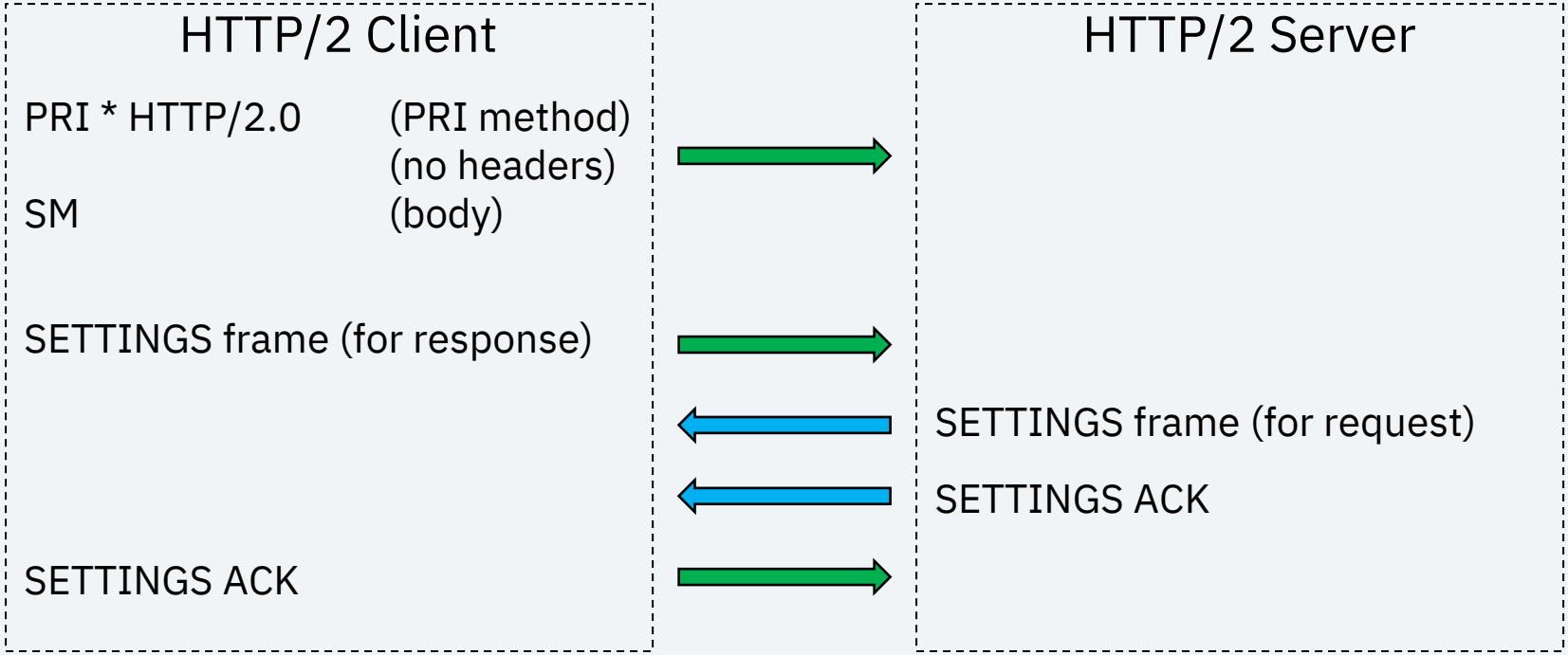
HTTP/1.x

- Text based format
- 1 socket = 1 concurrent request/reply
- Errors reported through status and status reason

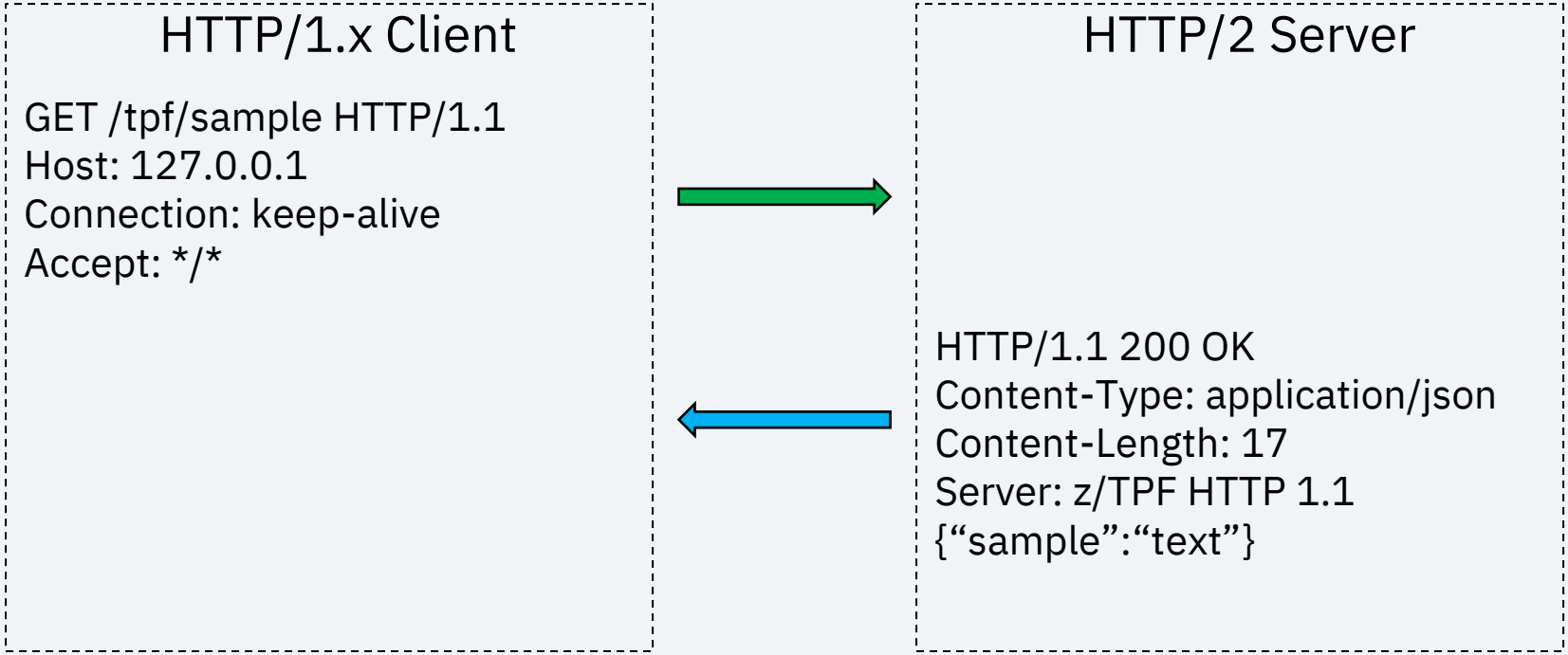
HTTP/2

- Frame based format
- 1 socket = n concurrent requests/replies (each request/reply pair uses a unique stream identifier)
- Errors reported through either status or stream error or connection error

Establishing an HTTP/2 Connection
HTTP/1.x Compatible Preface



HTTP/1.x Compatibility



HTTP/2 Message Format

A request/response consists of the following frames:

- **HEADERS** frame (pseudo headers and normal headers are compressed using HPACK)
- **CONTINUATION** frames - if more than 1 HEADERS frame needed
- **DATA** frames (body) - optional
- **HEADERS** frame (trailing headers) - optional

Every frame consists of a 9-byte frame header with the following information:

- **Length of frame payload** (limited by SETTINGS)
- **Frame type** (DATA, HEADERS, SETTINGS, PING, and so on)
- **Flags** (END_HEADERS, END_STREAM, ACK, and so on)
- **Stream identifier** (always increasing, no reuse)

HTTP/1.x v HTTP/2 Request

HTTP/1.x

GET /tpf/sample HTTP/1.1

Host: 127.0.0.1

Connection: keep-alive

Accept: */*

HTTP/2

HEADERS frame: (using HPACK)

:method: GET

:scheme: http

:path: /tpf/sample

:authority: 127.0.0.1

~~connection: keep-alive~~

accept: */*

HTTP/1.x v HTTP/2 Response

HTTP/1.x

HTTP/1.1 200 OK

Content-Type: application/json

Content-Length: 17

Server: z/TPF HTTP 1.1

{“sample”:“text”}

HTTP/2

HEADERS frame: (using HPACK)

:status: 200

status-reason: OK

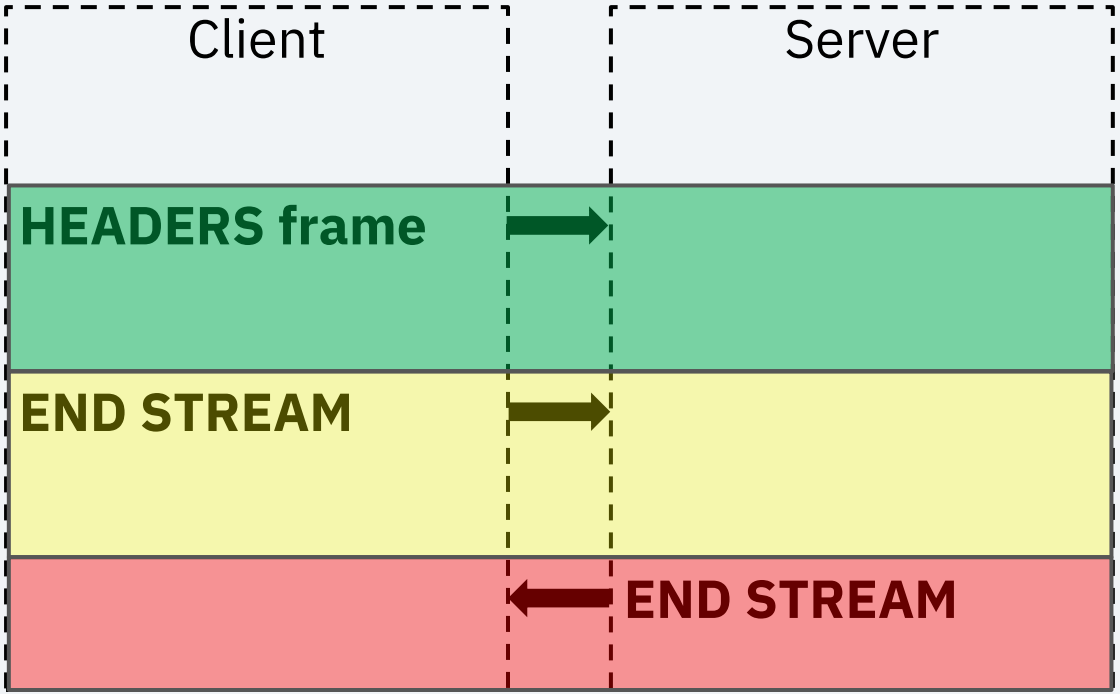
content-type: application/json

content-length: 17

server: z/TPF HTTP 2

DATA frame: {“sample”:”text”}

HTTP/2 Stream States



Stream State

open: both client and server can send/recv HTTP/2 message info

half-closed: only server can send HTTP/2 message info

closed: any remaining frames for the stream are discarded

HTTP/2 Concepts
Multiplexing

- Every frame contains a stream identifier (id 0 is reserved for connection management).
- Frame types and stream identifiers can be interleaved so that multiple requests or responses can be sent or received simultaneously.
- Exception: A HEADERS frame that doesn’t contain all the headers must use CONTINUATION frames to send the remainder of the headers without any other type of frame or stream identifier allowed until END_HEADERS is sent/received.

HTTP/2 Concepts
Flow Control

- Flow control to limit how much data can be received for either the whole connection or an individual stream.
- The HTTP body is sent in HTTP/2 through DATA frames. The window size defines the total amount of data that can be received by DATA frames.
- The initial window size for a connection and stream is defined as 64 KB until a SETTINGS frame or WINDOW_UPDATE frame is received.
- The SETTINGS_INITIAL_WINDOW_SIZE redefines the initial window size for all streams (active and new).
- When the window size for either the connection or stream is exhausted, no more DATA frames can be sent until a WINDOW_UPDATE is received.

z/TPF HTTP Server Changes

HTTP/1.x v HTTP/2 Application Interface
No application changes needed

HTTP/1.x	HTTP/2
tpf_httpsvr_req	*tpf_httpsvr_req*
requestVersion = HTTP_11	requestVersion = HTTP_2
tpf_httpsvr_token	*tpf_httpsvr_token*
locates connection info	locates connection + stream info
tpf_httpsvr_resp	*tpf_httpsvr_resp*
responseVersion = HTTP_11	responseVersion ignored
Header names are **mixed** case	Header names are **lower** case

z/TPF HTTP Server Multiplexing

- The z/TPF server will create an ECB for each new stream that is opened as was done for HTTP/1.x for each new request. This can result in multiple ECBs reading request data for a single connection.
- The z/TPF server will allow smaller responses to be interleaved in the middle of sending a large response to minimize impacting response times for those smaller response messages.
- If a stream becomes window blocked in sending a response, the remainder of the response will be stored in system heap (as determined by HTTP2WINBUF) to send when the window block is resolved and allow sending ECBs to exit.

z/TPF HTTP Server Flow Control

- The z/TPF server does not require any setting to handle flow control.
- The z/TPF server will set SETTINGS_INITIAL_WINDOW_SIZE to 2 GB-1 byte (the max) to avoid having to send a WINDOW_UPDATE for streams to send more data in a request.
- The z/TPF server will send a WINDOW_UPDATE at the start of a connection to change the connection window size to 2 GB-1 byte to avoid having to send too many WINDOW_UPDATE frames for receiving more data across multiple streams for the connection. A WINDOW_UPDATE will be sent after every 500 MB received for the connection (across all streams) to maintain a high connection window size.

z/TPF HTTP Server Settings

HTTP2–YES|NO

Enable or disable support for HTTP/2 connections. If you do not specify this parameter, HTTP/2 connections are not supported for this server.

MAXSTREAMS–numstreams

The maximum number of concurrent streams that are supported for an HTTP/2 connection. The default is 10.

HTTP2WINBUF–*size*

The size of the HTTP/2 connection send buffer to use when the response cannot be immediately sent to the client. This buffer is allocated from 64-bit system heap for each HTTP/2 connection. Specify the size in 4 KB units.

z/TPF HTTP Server Settings

PERSIST–YES|NO

Enable or disable persistent connections. This parameter affects only HTTP/1.x connections.

PERSIST_TIMEOUT–*numsecs*

The persistent connection timeout value, where *numsecs* is the number of seconds a client connection remains open without processing an inbound HTTP request. Valid values are 1-32767. When the timeout value is reached, the client connection is closed. This parameter also applies to HTTP/2 connections.

WSI_CONFORM–YES|NO

Enable or disable Organization for the Advancement of Structured Information Standards (OASIS) Web Services Interoperability (WS-I) conformance checking for SOAP request messages during communications binding request processing. z/TPF no longer supports SOAP; therefore, this parameter is ignored for HTTP/2 connections.

z/TPF HTTP Server Settings

MAXMSGSIZ – *maxmsgsiz*

The maximum allowed size, in bytes, of an HTTP request received by the server. The minimum *maxmsgsiz* value is 4096. HTTP request messages are saved in 31-bit ECB heap storage, so the maximum *maxmsgsiz* value cannot exceed the heap size. For HTTP/2 connections, the HTTP request message size is the total payload size of the HEADERS, CONTINUATION, and DATA frames minus any frame padding.

RCVBUF–*size*

The size of the HTTP server socket receive buffer. Specify larger values for the servers that support HTTP/2 because of the number of supported streams. Valid values are in the range 4096 - 1048576.

SNDBUF–*size*

The size of the HTTP server socket send buffer. Specify larger values for the servers that support HTTP/2 because of the number of supported streams. Valid values are in the range 4096 - 1048576.

HTTP/2 Server Display Statistics

ZHTPS STATs H2 [Server-sname (RESET) | ALL]

```
User:   ZHTPS STATS H2 S-HTPSV1

System: HTPS0014I 11:43:31 HTTP/2 CONNECTION STATISTICS FOR SERVER HTPSV1
SOCKET   MSTREAM ACTIVE HWACT  WINBUF USEBUF  HWBUF BLOCKED TIMEOUT LAST SID
-------- ------- ------ ------ ------ ------ ------ ------- ------- ----------
00C00009      20      7     15   400K    32K    62K       4       3      82937
00C00010      20      6     10   400K     0K     0K       0       0     429103
```

MSTREAM – HTTP server MAXSTREAMS value
WINBUF – HTTP server HTTP2WINBUF value
BLOCKED – number of streams that encountered window blocked condition
LAST SID – last stream id (opened), the client sends only odd-numbered stream ids

HTTP/2 Server Display Detail

```
User: ZHTPS DISP H2-C00009

System: HTPS0005I HTTP/2 DISPLAY FOR SOCKET C0001F
SOCKET SERVER     REMOTE IP:PORT       ACTIVE  SWINSIZE CWINSIZE DELAYQ
------ ---------- -------------------- ------- -------- -------- -------
C00009 HTTP81PORT 127.0.0.1:49152            7 2097152K 2096684K       0

HTTP/2 CONNECTION DETAILS
API TIME (MS):     AVG -      1.542  MAX -      18.203 AT 2025-01-31 13:03:15
RESP SIZE (BYTES): AVG -      32081  MAX -       70081 AT 2025-01-31 12:15:11
SEND QUEUE:      DEPTH -          0  MAX -           3 _
TCPIP WINDOW BLOCKED STREAM COUNT -          4  LAST - 2025-01-31 13:03:15
HTTP2 WINDOW BLOCKED STREAM COUNT -          0  LAST -
END OF DISPLAY
```

Changes to Existing HTTP Logging User Exit UHSR for HTTP/2

```
int                          version;          //Structure version
int                          type;             //response = 0, request = 1
int                          socket;           //Socket descriptor
int                          ssl;              //non-SSL = 0, SSL = 1
struct sockaddr_in           remote;           //remote IP/port
struct sockaddr_in           local;            //local IP/port
tpf_httpsvr_req             *req;              //HTTP server req, NULL if resp
tpf_httpsvr_resp            *resp;             //HTTP server resp, NULL if req
void                        *http_msg_format;  //Pointer to HTTP req/resp fmt (HTTP/2=NULL)
unsigned int                 http_msg_len;     //Length of HTTP req/response (HTTP/2=0)
char                         flag;             //flag byte
#define UHSR_RESP_SENT        0x80             //Response was sent
#define UHSR_CHUNKED_BODY     0x40             //Body in data was chunked
#define UHSR_RESP_COMP_BODY   0x20             //Body in data was compressed
char                         reserved_char[3]; //Reserved for IBM use only
unsigned int                 stream_id;        //HTTP/2 stream identifier
unsigned int                 error_code;       //HTTP/2 error code when resp is NULL
```


Changes to Existing HTTP Logging User Exit UHSR for HTTP/2

If the message format (http_msg_format, http_msg_len) is not referenced, no changes are required. Otherwise, use the following functions to create an HTTP/1.x message format for either HTTP/1.x or HTTP/2:

```
int tpf_httpFormatRequest(t_httpClientRequest *req,
                          char **http_msg_format,
                          char options);

int tpf_httpFormatResponse(tpf_httpsvr_resp *resp,
                           char **http_msg_format,
                           char options);

options: 0 or TPF_HTTP_FORMAT_NOBODY
```

Conclusion

PJ48022 (Apr 2025): z/TPF HTTP server support for HTTP/2

- This support reduces the number of sockets required when a large number of HTTP messages are exchanged between a remote client and the z/TPF system. No application changes are required.

Thank you