Proxy Pains and Benefits Supporting HTTP/2 End to End

Susan Hinrichs
shinrich@apache.org

September 22, 2021

Team Effort

- I am a Traffic Server Committer
  - This work was performed while at Yahoo
  - Recently moved to Aviatrix, so less Traffic Server for me
- Aaron Canary did some of this work while he was at Yahoo
- Brian Neradt has done work for testing HTTP/2 and is taking the testing torch at Yahoo
- Masakazu and Masaori have been involved in reviews and design discussions. Driving forces in HTTP/2 and HTTP/3 development in general

Motivation

- Traffic Server offers clients
  - HTTP/1.x (with and without TLS) and HTTP/2
- Traffic Server only offers origins HTTP/1.x (with and without TLS)
- Use cases for HTTP/2 to origin as well
  - Forward proxies – more transparent experience
  - Reverse proxies – take advantage of smart origins. Better session reuse
  - Proxy newer protocols like gRPC which are built over HTTP/2

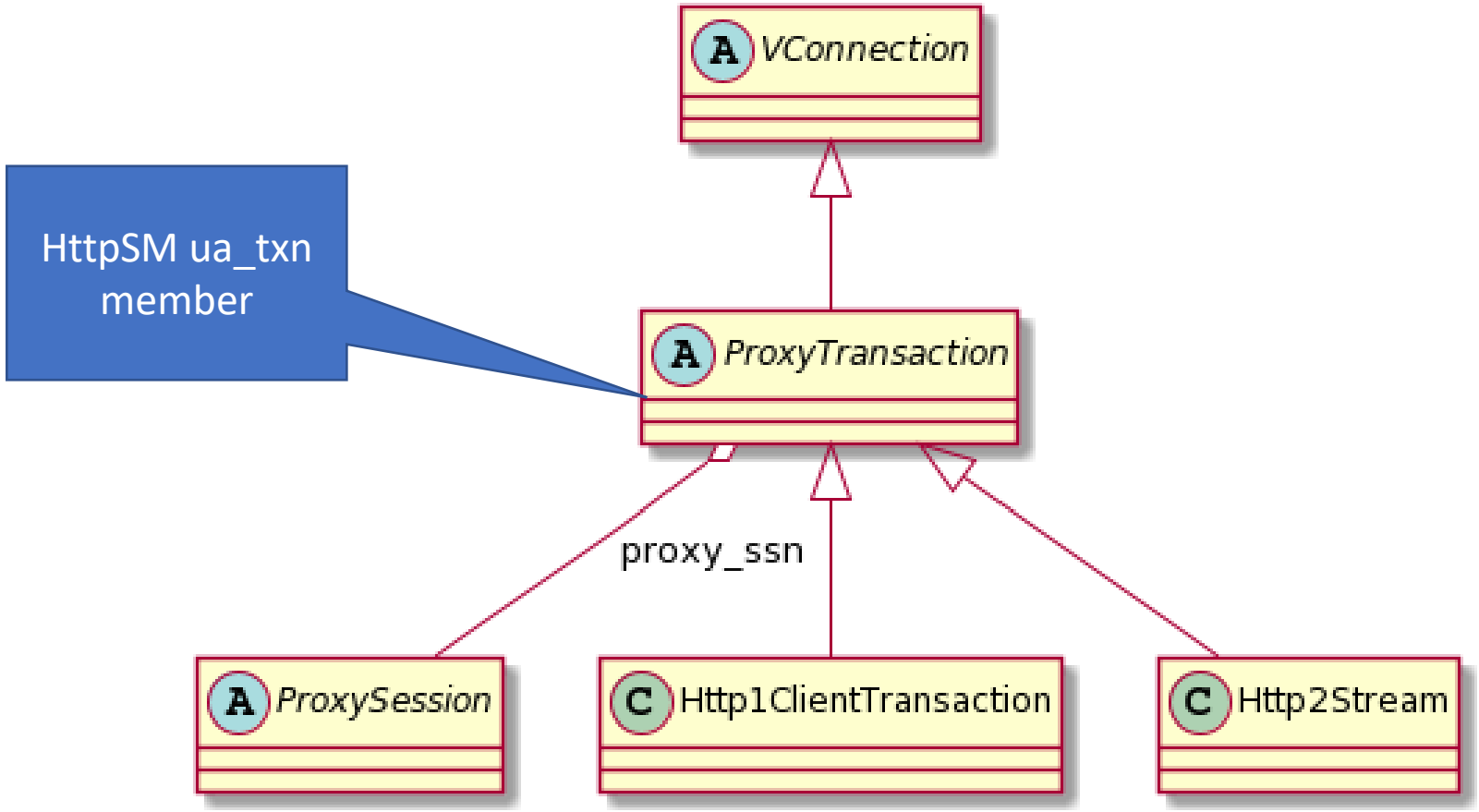
Short Traffic Server HTTP/2 History

- More than 8 years ago
  - ATS only supported HTTP/0.9,1.0,1.1 both inbound and outbound
- Around 8 years ago
  - ATS added support for SPDY inbound
  - Implemented as a plugin, wrapped with PluginVC
  - PluginVC very awkward to debug and performance tune
- Around 7 years ago
  - ATS added support for HTTP/2 inbound
  - Still PluginVC wrapped plugin

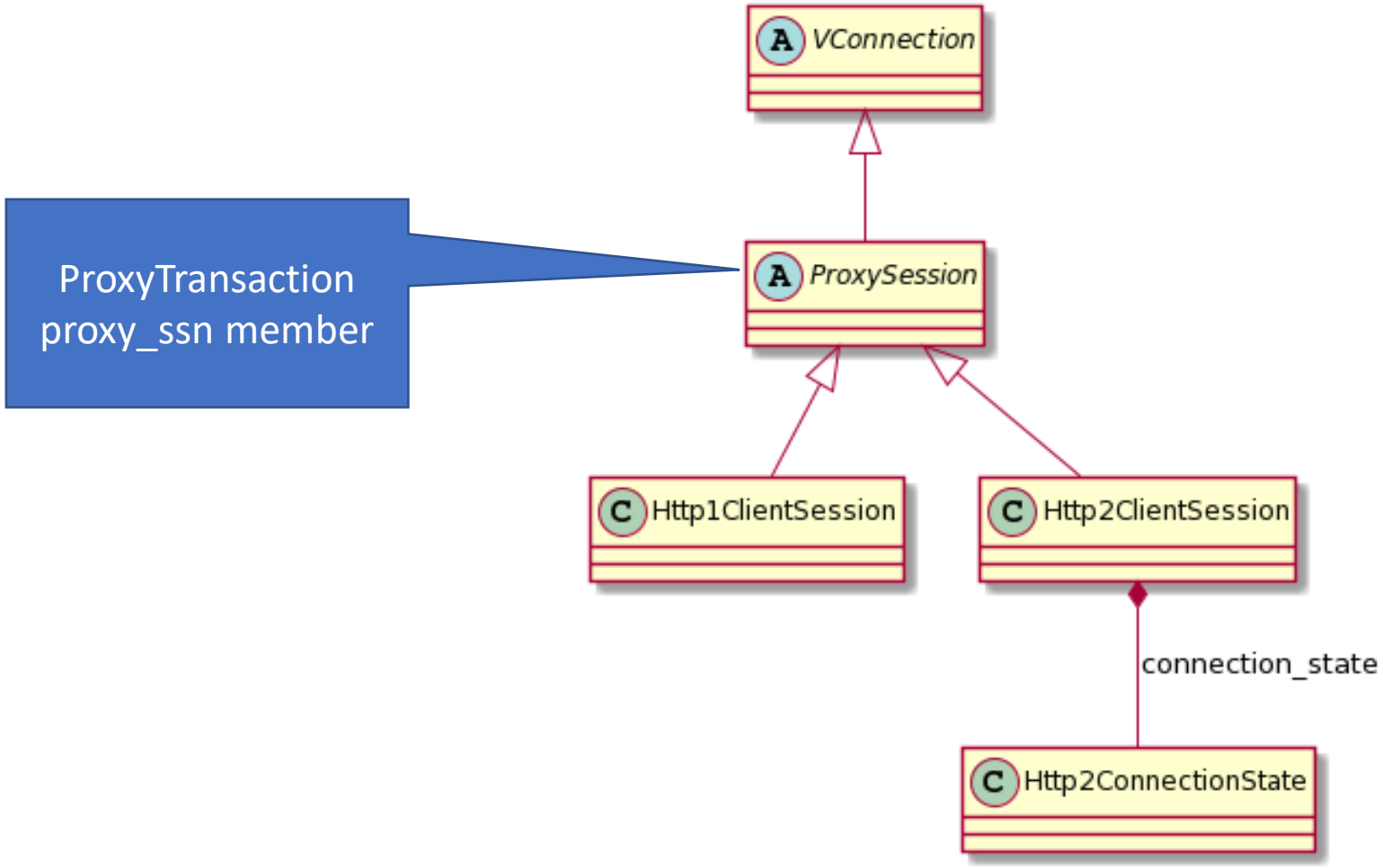
History Continued

- Around 6 years ago, I entered the scene
  - Implemented TS-3612 to separate details of HTTP transport from HTTP state machine (HttpSM).
  - Introduced ProxySession, ProxyTransaction
  - Landed in ATS 6.x
- Shortly after HTTP/2 moves from Plugin to core for inbound

Inbound Transaction Class Hierarchy



Inbound Session Class Hierarchy



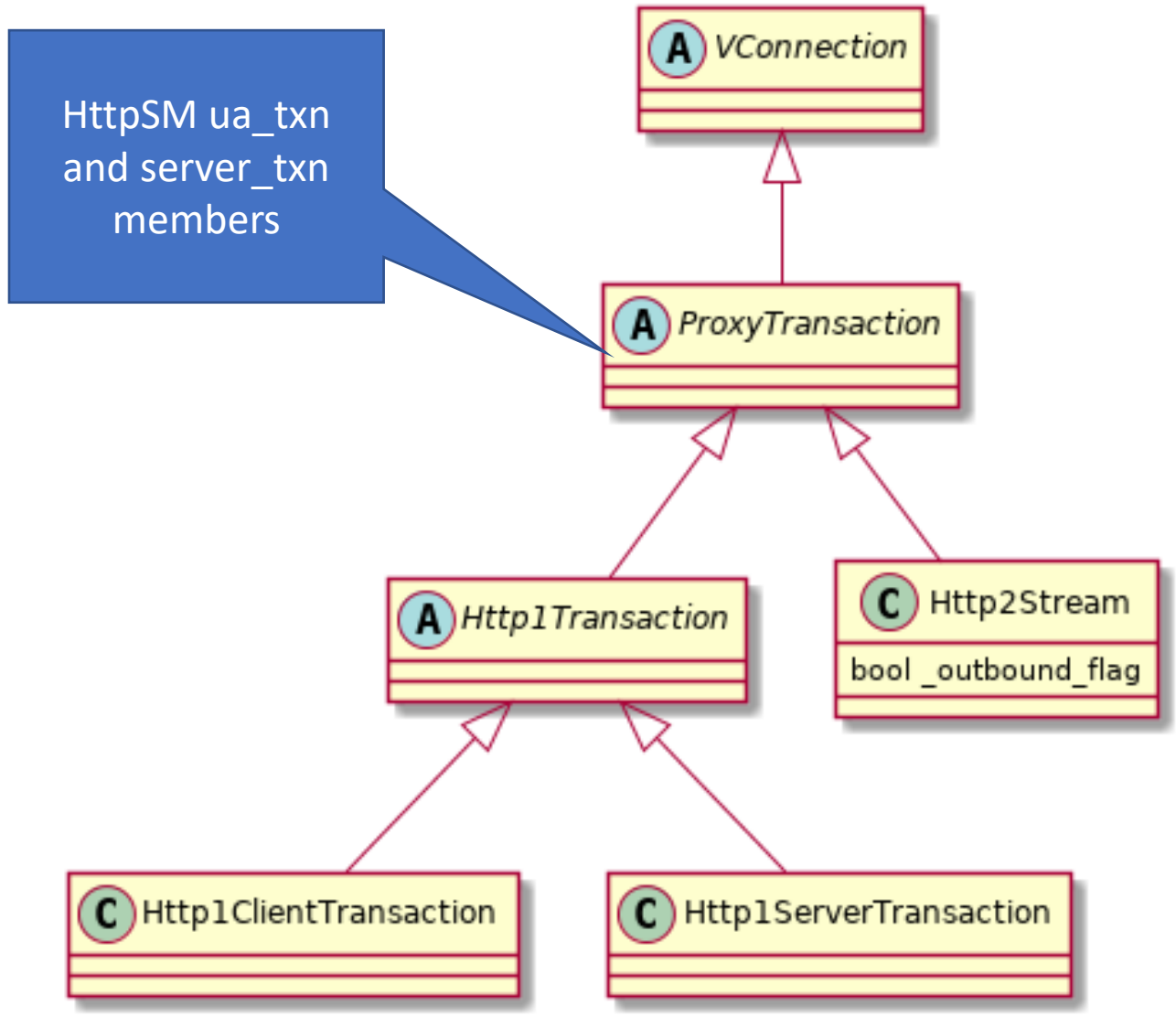
Outbound HTTP/2

- Some work started in 2018
  - I started prototyping HTTP/2 to origin
  - Kees Spoelstra had a proprietary implementation for a customer
  - Presentation at Cork Summit
- Then other projects took over for a bit, got back to HTTP/2 origin late 2020
  - Aaron Canary worked on abstracting Http1ServerSesson. In 9.1.
  - Set up HTTP/2 to origin PR March 2021. Marked for 10.0
  - Landed outbound protocol separation from HttpSM. Marked for 9.2.
  - Landed Http2Session class separation. Marked for 9.2

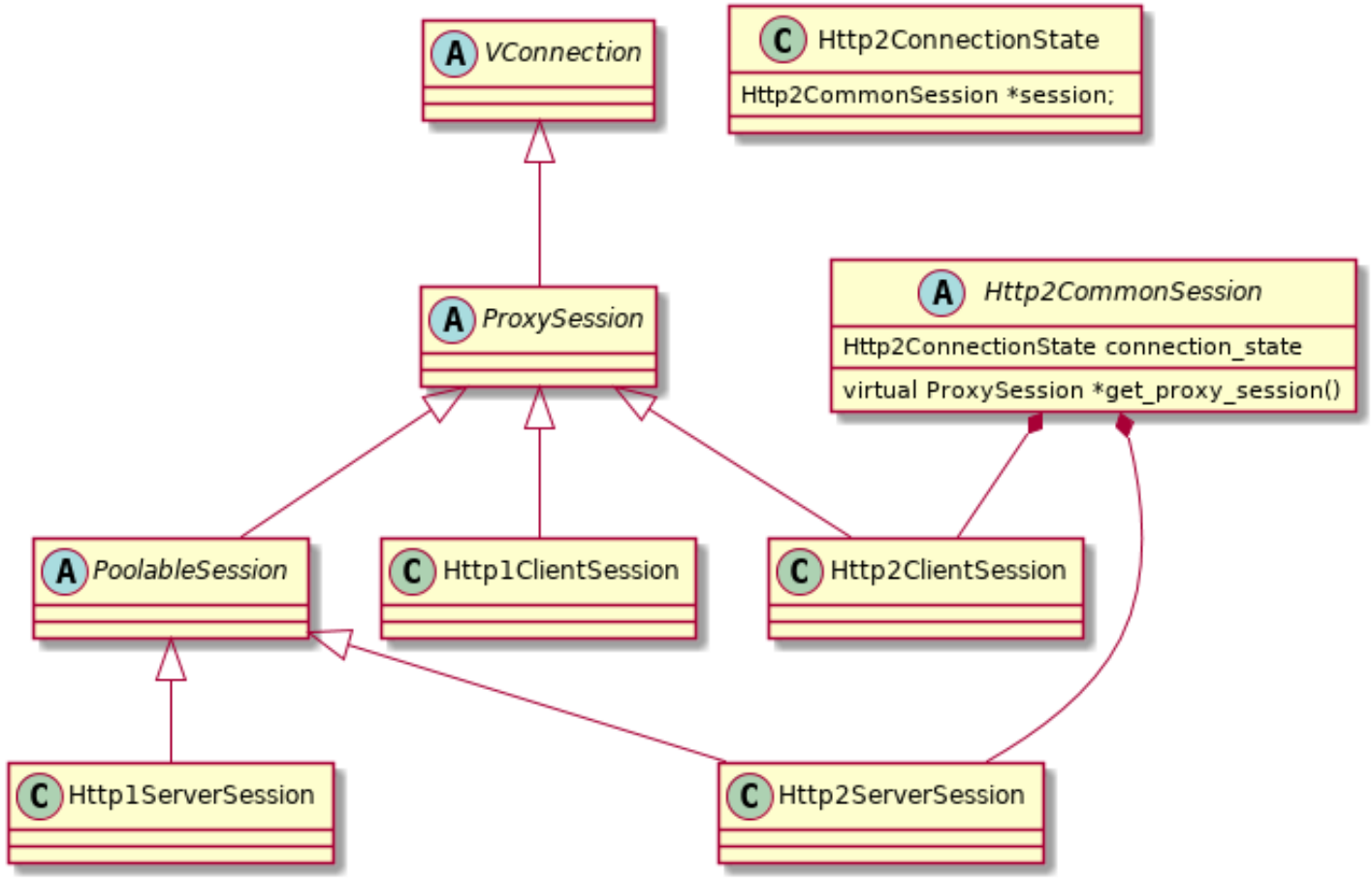
Current HTTP/2 Status

- Yahoo doing some production testing of a 9.1 branch
  - I was pushing this work, now Brian Neradt is
- Looking for other groups to work with this branch
- Aiming to land in 10.0, but ideally many of us get experience before then

Augmented ProxyTransaction Hierarchy



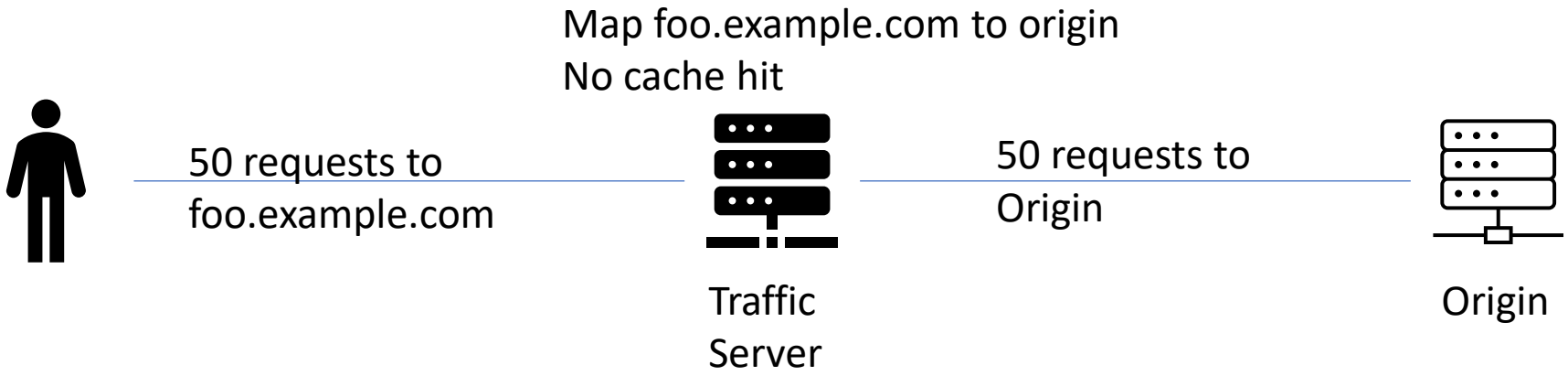
Augmented ProxySession Hierarchy



Configuring HTTP/2 as origin option

- ALPN setting proxy.config.ssl.client.alpn_protocols
  - By default empty (original HTTP/1.x behavior)
  - h2,http1.1 would offer h2 and http1.1 to origins
  - In our tests against clients, most offer h2,http1.1. Others offer combinations of spdy and older h2 pre-release versions
- Overridable
  - Can choose to offer HTTP/2 to only certain origins

Pooling while Connecting to Origins



- How do we handle connections to origin?
- If HTTP/1.1, fastest to make 50 simultaneous connection requests
  - Transaction requests for HTTP/1.1 are serialized over a connection
- If HTTP/2, probably better to set up 1 or 2 sessions and multiplex all transactions over those sessions

Adding Connecting_Pool

- The HTTP/2 branch adds a connecting_pool.
  - Look at hostdb to see the protocol negotiated on last connection
  - If it was HTTP/2, see if there is already a HTTP/2 session being negotiated
  - If so, add the current request to a queue for the existing request.
- When session is ready, signal all queued state machines

Origin Session Reuse

- Thanks to stream multiplexing, one HTTP/2 session can be the source for many simultaneous transactions
- Therefore, a HTTP/2 session cannot be moved between threads.
  - ATS asserts that all network IO operations for client and server side of a HTTP state machine occur on same thread
  - If a HTTP/2 session provided transactions for state machines on multiple threads, this assertion would be broken
- HTTP/2 origins cannot work with global origin session reuse pools
  - Must use hybrid or thread pools

Integration Testing

- The presence of an autest integration suite has greatly enhanced the stability of the HTTP/2 feature Branch
  - Compared to my experience making the similar inbound changes 6 years ago
  - The autest caught many stupid unintended side effects

ProxyVerifier

- Brian Neradt added HTTP/2 support for the server side of Proxy Verifier
- Used that to add some HTTP/2 to origin tests on the HTTP/2 Branch
  - H2origin.test.py and h2_origin_single_thread.test.py
  - Need more test cases always, but it is a start.

Production Testing

- Started limited production testing at the end of 2020
  - First phase just getting stability
  - After the first month, spent the time teasing out performance issues
- Testing in Yahoo Edge environment
  - Mix of caching (CDN) and pure proxying (ADN)
- Origins mix of Traffic Server, Nginx, Istio, and unknown

Production Performance Testing

- After the initial phase of fixing crashes, spent a lot of time analyzing performance issues
  - Mostly unexpected timeouts
  - Increase of ERR_CLIENT_ABORTS, ERR_CLIENT_READ, ERR_TIMEOUT
  - A GOAWAY from the origin would immediately shutdown all active streams
  - Protocol failures will have a far greater impact than they did on HTTP/1.x
- Tested in ADN/CDN Edge environment
- Fixed several general performance issues queued for 9.2
- Also identified some configuration changes

Abstract connection close header to avoid triggering H2 draining logic

- Fix landed in master and marked for 9.2
  - https://github.com/apache/trafficserver/pull/8178
  - Found while another Traffic Server was acting as origin. Should improve performance to end user as well
- The draining process for HTTP/2 was using the “Connection: close” header to signal that the HTTP/2 session should start shutting down
  - HTTP/2 should be ignoring the Connection headers
- HttpSM sets the Connection header to close on failures where the client or origin HTTP/1.1 connection is left in an unknown state
  - Assumed HTTP/2 was just ignoring all this
  - Instead the HTTP/2 origin would just randomly shutdown
  - Would finish existing streams, but limited the lifetime of the origin connection.

Dealing with origins returning early

- Process response headers before post tunnel is finished
  - https://github.com/apache/trafficserver/pull/7976
  - Was testing against an origin that would return 40x response when overloaded without reading the full post body.
- Pass through expect header and handle 100-continue response
  - https://github.com/apache/trafficserver/pull/7962
  - Also came out of testing during the early post return PR above

Delaying outbound stream id assignment

- Another issue was mis-ordering the id assignment for outbound streams
- For inbound streams, the ID is set when the stream is created. And we initially took that approach for outbound streams too
- However, there can be varying amounts of delay between the outbound stream creation and when the HEADER frame is sent to the origin.
  - It is a connection error if new streams do not have monotonically increasing IDs.
  - To avoid this, the current code separates the ID assignment from the stream object construction for outbound streams. The ID only gets set in the send_headers() method.

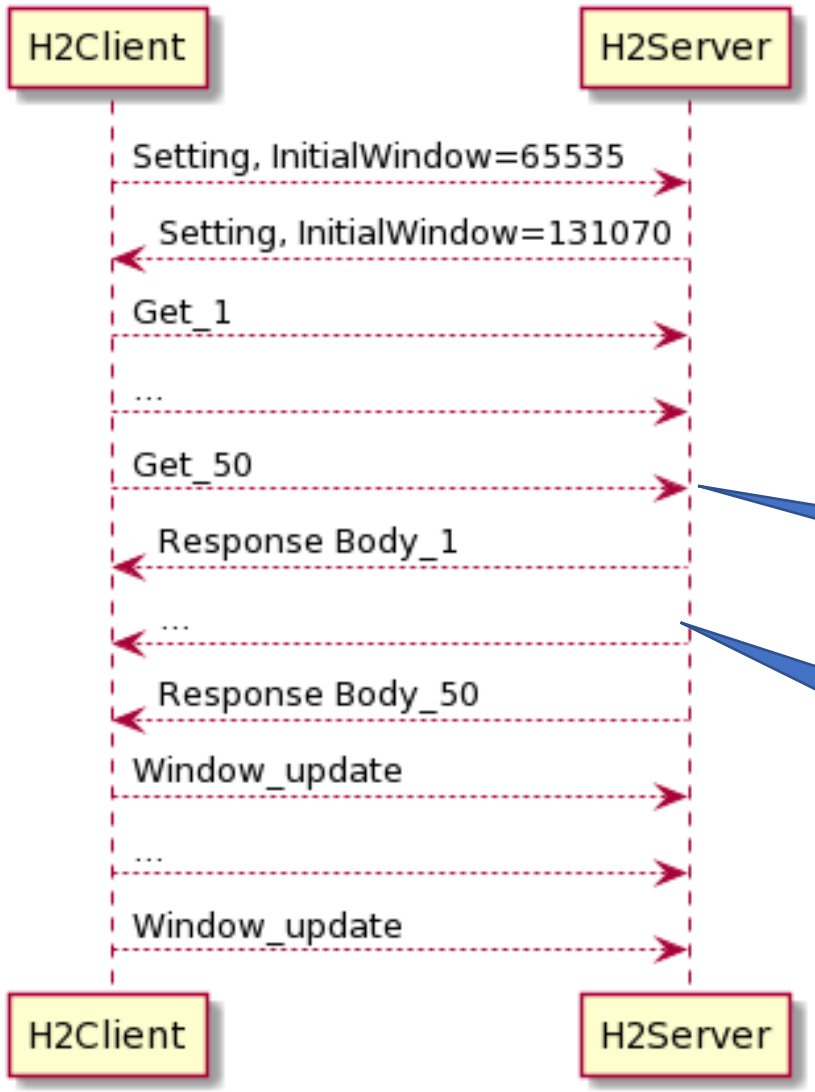
Reducing Max Concurrent Stream Overruns

- When ATS starts a H2 session with a peer, one of the settings is MAX_CONCURRENT_STREAMS (defaults to 100).
  - If ATS sends a HEADER frame to start a new stream and the peer thinks the limit has been reached, it will reject the new stream. Any DATA frames in play for the new stream will also fail and result in a connection error.
- The H2 to origin code tracks how many active streams it thinks the peer has. If it is at 90% of the limit, it will not create a new stream on that session. It will remove it from the session reuse pool.
- When the ATS stream count for its peer reaches 50% of the limit, ATS will add it back to the session reuse pool.

Performance Related Configuration Tuning

- Reducing keep alive origin session timeouts
  - Moving from global to hybrid pool means increase in total origin session counts
- Increase Http/2 window sizes
  - Debate on how to specify session versus window sizes in Issue #8199

HTTP/2 Data Windows



- Data Window for session and each stream
- Stop sending data on stream if either stream or session window are 0
- Traffic Server initializes both session and stream windows to the same initial window value.

Session_window=65535
stream1-50 window=65535

Some of the bodies will temporarily stall because the session window will go to 0 before the window_update frames appear

HTTP/2 Data Windows

- In my production testing, added a separate session window
  - PR #8203
- In Issue #8199
  - Masakazu proposes dynamically adjusting the per stream windows based on the number of active streams
- Could also make sure the current session is big enough to cover all concurrent streams and hope for the best
- For a busy, highly used HTTP/2 session the default is probably too small and causing unnecessary stalls

Preliminary Production Test Results

- Running two Traffic Server machines in the same POD
  - control 9.1
  - 9.1 + H2 to origin
- Comparing origin session reuse and TTMS (time to complete transaction) over 15 minute squid/access logs

Origin Reuse Comparison

9.1 Control

	Total Trans	Ave reuse	50%	80%	90%	95%	99%
H1 origin	2805337	7	3	14	21	29	47

9.1 + H2 enabled to 2 very active origins

	Total Trans	Ave Reuse	50%	80%	90%	95%	99%
H2 origin	367071	44497	52835	55138	55911	56366	56959
H1 origin	2460991	209	40	329	629	974	1895
Both	2828062	5957	69	610	50734	53716	56109

TTMS Comparison

9.1 Control

	Total Trans	Ave TTMS	50%	80%	90%	95%	99%
H1 origin	2805337	147	50	179	313	446	972

9.1 + H2 enabled to 2 very active origins

	Total Trans	Ave TTMS	50%	80%	90%	95%	99%
H2 origin	367071	142	36	62	111	184	1744
H1 origin	2460991	144	47	193	327	466	957
Both	2828062	144	42	173	312	447	987

Proxying gRPC

- gRPC requires HTTP/2 on both sides
- gPRC requires trailing headers
- Traffic Server currently parses and ignores trailing headers

Proxying gRPC

- The HTTP/2 Branch adds some support for trailing headers and gRPC
  - Trailing HTTP/2 header frames only. Not HTTP/1.1 chunked tailing headers.
  - I tested some of the basic gRPC Python examples through Traffic Server.

    The most recent test was late 2020
- In addition, good gRPC support may require origin connections dedicated to clients
  - Traffic Server offers proxy.config.http.attach_server_session_to_client
  - Need to test that scenario

Future Changes

- Of course this is not the end, there will be many more issues moving forward.

Reduce Header Serialization

- Currently HTTP/2 logic serializes headers. HttpSM, parses them back to headers. HttpSM serializes headers back to the ProxyTransaction.
  - If both sides are HTTP/2, ATS is serializing the headers 3 times.
  - Being tracked by issue #5230

Support QUIC and HTTP/3 outbound

- With this infrastructure in place, the time between supporting QUIC inbound and QUIC outbound should not be so long
- And whatever the next protocol that comes along.

Any questions?

- Please go test HTTP/2 to origin!
- shinrich@apache.org