State of Http/2 to Origin

Susan Hinrichs and Kees Spoelstra

ATS Euro Tour

Cork

May 2018

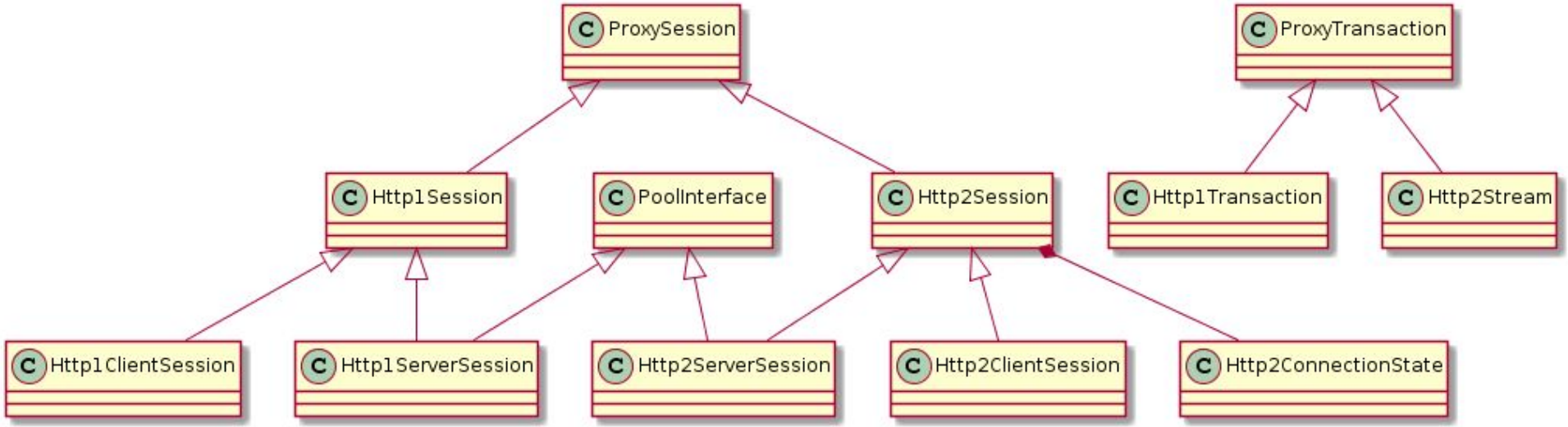
Traffic to Origin

- Currently only HTTP/1.x over TCP or TCP/TLS.
- Now supporting HTTP/2 from client (soon QUIC as well)
- Would be nice to support these protocols end-to-end
- Use cases
  - Forward proxies
    - More transparent experience
  - Reverse proxies
    - Take advantage of smart origins
  - Newer protocols that leverage HTTP/2
    - Like gRPC https://grpc.io/

HTTP/2 to Origin

- Github project page
  - https://github.com/apache/trafficserver/projects/9
- Susan’s branch
  - https://github.com/shinrich/trafficserver/pull/1
  - Forked last December
  - Basic connectivity to origin
  - Very basic grpc functions operational
    - Much effort on chunked encoding and HTTP/2
- Kees
  - Private project leveraging nghttp2 to get HTTP/2 to origin working

HTTP/2 Class Diagram



What does server session reuse mean with HTTP/2?

- My original thought was that HTTP/2 sessions could be source of many transactions. Need not take HTTP/2 session out of table after creating a stream
  - Global Pool makes no sense then. Cannot have one HTTP/2 Session with two active streams running on two different threads
- Reasonable view from a reverse proxy perspective.
  - One client HTTP/2 session should be pinned to the one HTTP/2 session used by origin
  - Smart origin may be expecting this (for scheduling and prioritization and pushing)
  - Could extend the existing server session pinning/binding logic.
  - Connection reuse is per client in this case.
  - Also required for the grpc scenario
  - HOL has only impact on one client (except with thundering herd)
  - Mandatory in some forward proxy cases,End to end HTTPS connections are often considered private.

Connection/Stream Queuing

- Issue identified by Kees
- Client may connect and make x stream requests before connection to origin has been established
  - Naive approach (what I have implemented currently) would independently start setting up 10 H2 connections to the same origin.
  - Instead should queue stream requests when a H2 connection to origin is in progress.
- Do we need a safe mode?
  - Queue until we know max_concurrent_streams from origin, spec says at least a high number, but nodejs servers might do otherwise.
- Do we want to propagate max_concurrent_streams from origin to the client?

Connection chaining

- An H2 origin server might decide to stop accepting streams for several reasons.
  - It can still send bytes on streams which are currently active.
  - This means that we need to attach multiple sessions to the client session
  - This has been solved in one implementation by chaining the sessions

Priority forwarding

- In the forward proxy case we identified unwanted behaviour (google image search)
  - 50 streams were opened at the same time
  - Because proxy to origin did not support priority it just evened out all streams
  - Which makes for a terrible user experience
- Client to proxy priority helped a little bit
- Is there a smart way we can forward the priority tree or a translation to the proxy to origin

  Challenges:
  - Streams will not arrive in same order
  - Some streams will be served from cache
  - Client to proxy connections can be split over multiple proxy to origin connections
  - The above all result in a completely different stream configuration for proxy to origin

Chunked Encoding and HTTP/2

- GRPC relies on chunked encoding on POST bodies and GET/POST responses
  - Our existing H2 support for POST chunked encodings was non-existent
  - Naive implementation (my current branch) will copy H2 frames with chunked encoding header set to traditional chunked encoding buffer for state machine processing then copy back to individual frames on the other side
    - Eventually may want to understand both sides are H2 and have an alternative form to pass the data through
- Also uses chunked encoding trailer headers
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Trailer

Control which protocols ATS advertises upstream

- Proxy.config.ssl.client.alpn_protocols - list protocols ATS is willing to accept
- Per origin?

1:1 protocol fwd proxy case

- HTTPS often considered private, mirroring connection properties as much as possible
- H2 to H1 private is challenging, P-O connection pool per C-P connection?
- H1 to H2 is ok, but cryptographically maybe not
- Not elaborating too much, but involves:
  - Probing the origin, saving max protocol
  - More control over connection
    - attaching data (we need to finish PRs)
    - lifecycle events (we need to finish PRs)
    - A connection is not a session :)
  - Lots of SSL stuff
    - dropping back to H1 in ALPN
    - more callbacks from openssl
    - Session reuse bypasses a lot of callbacks :(
    - fun stuff
- Also applies to reverse proxy with uncontrolled origins

Next steps

- I need to catch up my branch - Many changes to master since December
- Do we have all the use cases?
  - Find some more realistic grpc scenarios to exercise
- Make autests
  - Need H2 capable microserver or other test origin
- Make client/server session pinning work
- Disable global pools and H2 origin connections
- Make queued H2 streams work.
- Make sure this lines up with how QUIC is progressing.

Comments

Maskit notes that we need to preserve the never indexed fields
https://tools.ietf.org/html/rfc7541#section-6.2.3