QMux

comparison of wire protocols proposed

IETF 123 Madrid

Kazuho Oku, Lucas Pardue

Charter requirements of a wire protocol

- charter patch awaiting IESG review
  - **not** seeking to adopt a protocol until approved
- June interim meeting identified a broad solution space
  - let's explore that to establish common understanding
- abridged requirements of charter patch
  - reliable and bidirectional byte stream substrate
  - when substrate is insecure, TLS will be the default security provider
  - substrate must provide congestion-management capabilities

4 approaches in a nutshell

	frame encoding	negotiation	establishment latency*	record layer
a) WT-over-H1	capsules	HTTP header	3 RT	TLS
b) capsules	capsules	QUIC v1 TP?	2 RT	TLS
c) v1 frames	QUIC v1	QUIC v1 TP	2 RT	TLS
d) v1 frames & packets	QUIC v1	QUIC v1 TP	2 RT	QUIC v1-ish
(QUIC v1)	QUIC v1	QUIC v1 TP	1 RT	QUIC v1

* measured as the time between when the client sends the first packet of the connection (e.g., TCP SYN) and when it sends the first flight of application data, while doing a full handshake

a) WT-over-H1

WT-over-H1: overview

- ports WebTransport (WT) over HTTP/2 to HTTP/1.1
- avoids H2 complexity useless for QMux; e.g., HPACK
- avoids putting QUIC streams inside H2 streams, removing redundancy (no nesting of frames or flow control)
- some QUIC extensions are incorporated into WT
  - DATAGRAMs
  - RESET_STREAM always has the Reliable Size field

WT-over-H1: frame encoding

- uses capsules:
  - a type-length-value encoding defined by HTTP WG
  - unknown capsule types are silently dropped
  - WT-over-H2 defines how each QUIC frame is encoded as capsules
  - frames that can be self-delimiting (e.g., MAX_DATA) will have a length field

WT-over-H1: frame encoding (cont'd)

- takes advantage of the underlying substrate being reliable and in-order, e.g.:
  - STREAM capsules omit the offset field, as the receiver can determine the offset
  - sending RESET_STREAM capsules for a closed stream is prohibited
  - sending multiple RESET_STREAM capsules for the same stream is prohibited

WT-over-H1: decoding frames

- endpoints may receive partial capsules, as the underlying substrate is byte-oriented
- when receiving a partial capsule other than STREAM, receivers buffer and wait until a complete capsule is received
    - this is done in a type-agnostic manner, as all capsules have the length field right after the type field
- after receiving a complete capsule, receivers parse the capsule payload, simultaneously validating if the payload has not been truncated (or has excess bytes at the end)

WT-over-H1: decoding frames (cont'd)

- for STREAM capsules, receivers stream the payload of a STREAM capsule to the application, before entire capsule is received
  - this is due to the size of capsules being unbounded

WT-over-H1: establishment latency

- establishment latency is 3 RTs
  - 1st RT: TCP SYN exchange
  - 2nd RT: TLS handshake
  - 3rd RT: client sends HTTP request, and the server's HTTP response contains the initial flow control credits that allows the client to send its first flight of application data
- note: WT-over-H2 requires one less RT by providing servers the option to send the initial credits using H2 SETTINGS conveyed by TLS 0.5 RTT data, but H1 does not have SETTINGS

WT-over-H1: record layer

- when the underlying substrate is insecure (e.g., TCP), TLS is used and WT-over-H1 runs on top of TLS
- when the underlying substrate is secure (e.g., UNIX sockets), WV-over-H1 runs directly on the connection
    - i.e., no encryption above the transport layer

b) capsules

capsules: overview

- similar to WT-over-H1, but avoids the use of HTTP request-response cycle
  - once the handshake of the underlying substrate is complete, each endpoint sends the Transport Parameters (initial flow control credits) immediately

capsules: frame encoding / decoding

- same as WT-over-H1

capsules: establishment latency

- 2 RTs
  - 1st RT: TCP SYN exchange
  - 2nd RT: TLS handshake & server sends initial flow control credits

capsules: record layer

- same as WT-over-H1

capsules: uncertainties

- do we want to depend on the capsule registry managed by HTTP WG, when not using HTTP?
- do we want to retain the divergence of WT-over-H1/H2 e.g.:
    - dropping unknown capsules, as opposed to use of unknown frames being a protocol violation in QUIC v1?
    - tightening the stream state machinery, forbidding sending out-of-date / redundant frames?

c) v1 frames

(the one defined in draft-kazuho-quic-on-streams-00, which is now draft-opik-quic-qmux)

v1 frames: overview

- the idea: send QUIC v1 frames directly on top of a reliable and in-order substrate, while retaining QUIC v1 as much as possible
- uses QUIC v1 frames and stream state machinery as-is
- the first frame sent by each endpoint carries the Transport Parameters

v1 frames: frame encoding

- same as QUIC v1
  - the length of STREAM frames without an explicit length field is determined by a new Transport Parameter called max_frame_size
  - the default of max_frame_size is 16KB (i.e., the default maximum size of a TLS record)
- for each stream, STREAM frames are always sent in order
  - no need for buffering or reassembly
  - i.e., the offset field of STREAM frames is a pure redundancy

v1 frames: decoding frames

- endpoints may receive partial frames, as the underlying substrate is byte-oriented
- when receiving new data, receivers call the QUIC v1 frame decoder
    - if the frame is truncated, receivers retain the input (as opposed to generating errors in QUIC v1)
    - when more bytes are received, receivers repeat this process of speculative decoding, until a complete frame is received
- no need for a dedicated logic to handle frame truncation

v1 frames: decoding frames (cont'd)

- as an optimization, receivers might stream the payload of STREAM frames to the application before a complete STREAM frame is received
  - there's little reason to do so when TLS is used and max_frame_size is set to 16KB, because if a STREAM frame fits into a single TLS record, the complete STREAM frame would always become available at once

v1 frames: establishment latency

- same as capsules

v1 frames: record layer

- same as WT-over-H1, capsules

v1 frames: uncertainties

- STREAM frame encoding:
  - do we want to omit the offset field?
  - do we want to prohibit the use of STREAM frames without an explicit length field?

d) v1 frames & packets

v1 frames & packets: overview

- the idea: send QUIC packets over the underlying substrate
  - encryption is done by QMux (as opposed to relying on TLS as the underlying substrate)
- for each stream, STREAM frames are always sent in order
  - no need for buffering or reassembly
  - i.e., offset field of STREAM frames are a pure redundancy

v1 frames & packets: frame encoding

- wire encoding same as QUIC v1
- never receives partial frames, as frames never cross the “packet” boundary
    - in other words, no need for the “streaming” processing of STREAM frames

v1 frames & packets: packet encoding

- because the underlying substrate is byte-oriented:
  - endpoints may receive partial packets and buffer them
  - all the “packet” formats need a length field
    - i.e., the packet format of QUIC v1 cannot be used as-is, as the short header packet does not have a length field
- CID (and packet numbers) are unneeded

v1 frames & packets: establishment latency

- same as capsules, v1 frames

v1 frames & packets: record layer

- when the underlying substrate is insecure (e.g., TCP), QMux runs directly on top of that substrate, encrypting the data by itself
- deployments using secure substrates should either:
  - pay the cost of performing AEAD encryption for no benefit, or
  - a separate specification is needed for these deployments
    - note: TLS/1.3 nor QUIC v1 defines a null-null-null ciphersuite

v1 frames & packets: record layer (cont’d)

- encryption offload might not be available, because the AEAD structure is neither TLS nor QUIC-v1-over-UDP

v1 frames & packets: uncertainties

- need to agree on how to tweak the packet format
    - how to add length field to short packets
    - if CID and packet numbers should be omitted
- needs a separate design for deployments using secure substrates

complexity depends on perspective

complexity depends on perspective

	complexity*
coding from scratch**	b ~ c < a < d
coding on top of QUIC v1 stacks***	c ~ d < b < a
coding on top of WT-over-H2 stacks****	a < b < c < d
specifying the protocol (“uncertainties”)	a < c < b < d

a) WT-over-H1, b) capsules, c) v1 frames, d) v1 frames & packets

*: the leftmost one is the least complex
**: *a* needs HTTP; *d* requires implementing the packetization layer (incl. AEAD)
***: all require buffering of incomplete capsules, frames, or packets; *b* uses different frame encoding and tightens the stream state machinery; *a* requires use of HTTP in addition
****: *a* and *b* are derivations of WT-over-H2, *c* and *d* are not; *d* requires implementing packetization

open questions

- frame encoding: capsules or QUIC v1?
- how much do we want to diverge from QUIC v1?
  - omit the offset field of STREAM frames
  - tighten stream state machinery to prevent use of out-of-date / redundant frames
- rely on TLS, or port the packetization layer of QUIC v1?