ALTO New Transport using HTTP/2

draft-ietf-alto-new-transport-01

Roland Schott
Y. Richard Yang
Kai Gao
Jensen Zhang

July 26, 2022

IETF 114

Outline

- Document activities
- High-level recap
- Major changes from IETF 113
- Discussions and remaining issues to finalize

Document Activities

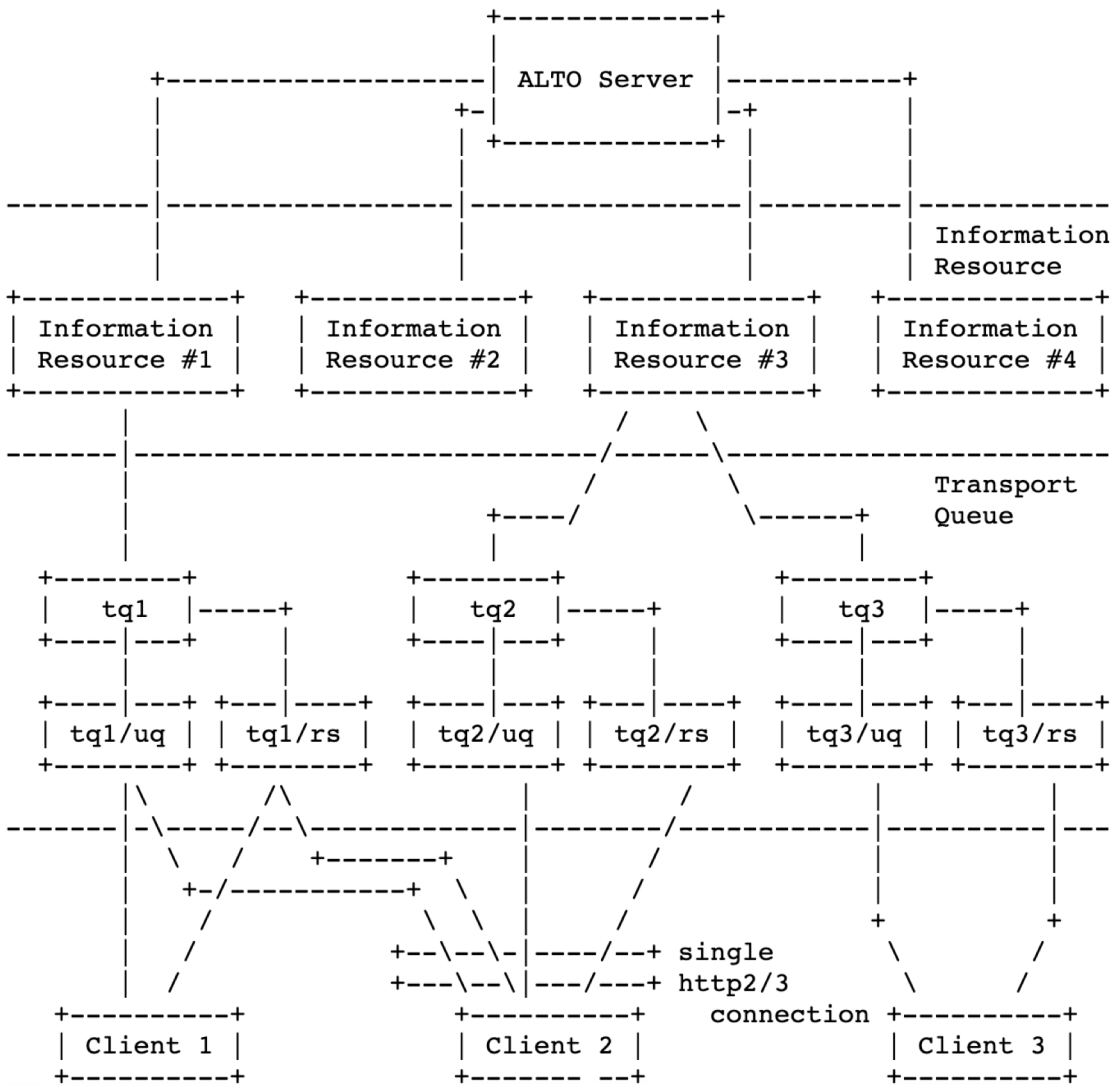
- Adopted as working group (WG) document
  - draft-ietf-alto-new-transport-00 submitted on June 22, 2022
- Updates to fix review feedbacks
  - Many helpful feedback from WG chairs (Med and Qin) and members (Luis, Sabine, Jordi)
  - draft-ietf-alto-new-transport-01 submitted on July 10, 2022
- Three excellent, early HTTP expert reviews
  - [MT] Martin Thompson (July 11)
    - https://mailarchive.ietf.org/arch/msg/alto/sa1Pv7jmTfBF3TbGuJr_PffIjXg/
  - [SD] Spencer Dawkins (July 15)
    - https://datatracker.ietf.org/doc/review-ietf-alto-new-transport-01-artart-early-dawkins-2022-07-15/
  - [MN] Mark Nottingham (July 17)
    - https://mailarchive.ietf.org/arch/msg/alto/D84S0qLbgtpL0-jf93gNPS3NUJE/
  - More discussion details see WG mailing list archive

Recap: ALTO New Transport Design Requirements

- From ALTO base protocol [RFC 7285]
  - R0: Client can request any ALTO resource using the connection, just as using ALTO base protocol using HTTP/1.x
- From ALTO SSE [RFC 8895]
  - R1: Client can request the addition (start) of incremental updates to a resource
  - R2: Client can request the deletion (stop) of incremental updates to a resource
  - R3: Server can signal to the client the start or stop of incremental updates to a resource
  - R4: Server can choose the type of each incremental update encoding, as long as the type is indicated to be acceptable by the client
- From ALTO base framework [RFC 7285]
  - R5: Design follows basic HTTP Representational State Transfer architecture if possible
    - Can use only a limited number of verbs (GET, POST, PUT, DELETE, HEAD)
  - R6: Design takes advantage of HTTP/2 design features such as parallel transfer and respects HTTP/2 semantics [PUSH_PROMISE]
- Allow flexible deployment
  - R7: Capability negotiation

Recap: New Transport Architecture

- An ALTO server provides multiple info resources {R[i]}
  - URIs announced in Information Resource Directory (IRD)
- An ALTO client uses a single HTTP/[2-3] connection to receive the content of multiple resources
- Content of a resource with updates stored in an update queue (URI)
- Client can pull items in the update queue from server or server can push items in the update queue to client



Major Changes from IETF 113

- Substantial updates to specify protocol details

Discussions and Remaining Issues

- Excellent comments and discussions from HTTP experts
  - Main remaining issues for the WG are to finalize the concurrency control and semantics of the transport of the information items
  - They may represent generic problem for HTTP/[2-3] design
  - The finalization of the remaining issues does not look like to need substantial texts, but need careful WG decision and specification

Discuss 1: Finalizing Op Mode: Client Pull/Server Push/ Client Long Pull/Server Put

- Current mechanisms
  - Client pull

```
Client opens a connection to the server
Client opens/identifies a transport queue tq
  // pull mode
  Client requests transport queue status of tq
  Client requests an element in the incremental update queue

  // push mode
  Client becomes a receiver
  Client receives incremental push updates
Client closes the transport queue tq
Client closes the connection
```

Figure 3: ALTO New Transport Information Structure.

```
{
  [
    {"seq":        101,
     "media-type": "application/alto-costmap+json",
     "tag":        "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe" },
    {"seq":        102,
     "media-type": "application/merge-patch+json",
     "tag":        "cdf0222x59740b0b2e3f8eb1d4785acd42231bfe" },
    {"seq":        103,
     "media-type": "application/merge-patch+json",
     "tag":        "8eb1d4785acd42231bfecdf0222x59740b0b2e3f",
     "link":       "/tqs/2718281828459/snapshot/2e3f"}

  ],
  HEADERS
    + END_STREAM
    + END_HEADERS
      :method = GET
      :scheme = https
      :path = /tqs/2718281828459/uq/101
      host = alto.example.com
      accept = application/alto-error+json,
               application/alto-costmap+json
```

Discuss 1: Finalizing Op Mode: Client Pull/Server Push/ Client Long Pull/Server Put

- Current mechanisms
  - Client pull
  - Server push (push_promise)

```
Client opens a connection to the server
Client opens/identifies a transport queue tq
  // pull mode
  Client requests transport queue status of tq
  Client requests an element in the incremental update queue

  // push mode
  Client becomes a receiver
  Client receives incremental push updates
Client closes the transport queue tq
Client closes the connection
```

Figure 3: ALTO New Transport Information Structure.

```
Server -> client PUSH_PROMISE in current stream:

PUSH_PROMISE
  - END_STREAM
    Promised Stream 4
    HEADER BLOCK
    :method = GET
    :scheme = https
    :path = /tqs/2718281828459/uq/101
    host = alto.example.com
    accept = application/alto-error+json,
                 application/alto-costmap+json

Server -> client content Stream 4:

HEADERS
  + END_STREAM
  + END_HEADERS
    :status = 200
    content-type = application/alto-costmap+json
    content-length = TBD
```

Discuss 1: Finalizing Op Mode: Client Pull/Server Push/ Client Long Pull/Server Put

- Incremental push alternatives [MN review]
  - Client long pull
    - Allow request on next seq

```
Client opens a connection to the server
Client opens/identifies a transport queue tq
  // pull mode
  Client requests transport queue status of tq
  Client requests an element in the incremental update queue

  // push mode
  Client becomes a receiver
  Client receives incremental push updates
Client closes the transport queue tq
Client closes the connection
```

Figure 3: ALTO New Transport Information Structure.

```
{
  [
    {"seq":        101,
     "media-type": "application/alto-costmap+json",
     "tag":        "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe" },
    {"seq":        102,
     "media-type": "application/merge-patch+json",
     "tag":        "cdf0222x59740b0b2e3f8eb1d4785acd42231bfe" },
    {"seq":        103,
     "media-type": "application/merge-patch+json",
     "tag":        "8eb1d4785acd42231bfecdf0222x59740b0b2e3f",
     "link":       "/tqs/2718281828459/snapshot/2e3f"}

  ],
  HEADERS
    + END_STREAM
    + END_HEADERS
      :method = GET
      :scheme = https
      :path = /tqs/2718281828459/uq/104
      host = alto.example.com
      accept = application/alto-error+json,
               application/alto-costmap+json
```

Discuss 1: Finalizing Op Mode: Client Pull/Server Push/ Client Long Pull/Server Put

- Incremental push alternatives
  - Client long pull
  - Server **put [MN review]**
    - Benefit:
      - Avoid “awkward” promise (current spec: MUST NOT cancel push promise)
    - Issue: semantics
      - Client conceptually is only a cache, not a persistent state replica

```
Server -> client PUSH_PROMISE in current stream:

~~PUSH_PROMISE~~
  - END_STREAM
  ~~Promised Stream 4~~
    HEADER BLOCK
    :method = ~~GET~~ PUT
    :scheme = https
    :path = /tqs/2718281828459/uq/101
    host = alto.example.com
    accept = application/alto-error+json,
                 application/alto-costmap+json

Server -> client content Stream 4:

HEADERS
  + END_STREAM
  + END_HEADERS
    :status = 200
    content-type = application/alto-costmap+json
    content-length = TBD
```

Discuss 2: How Much to Specify Ordering Control: Transport-Aware of App Semantics [MN, MT, SD reviews]

Design 1:

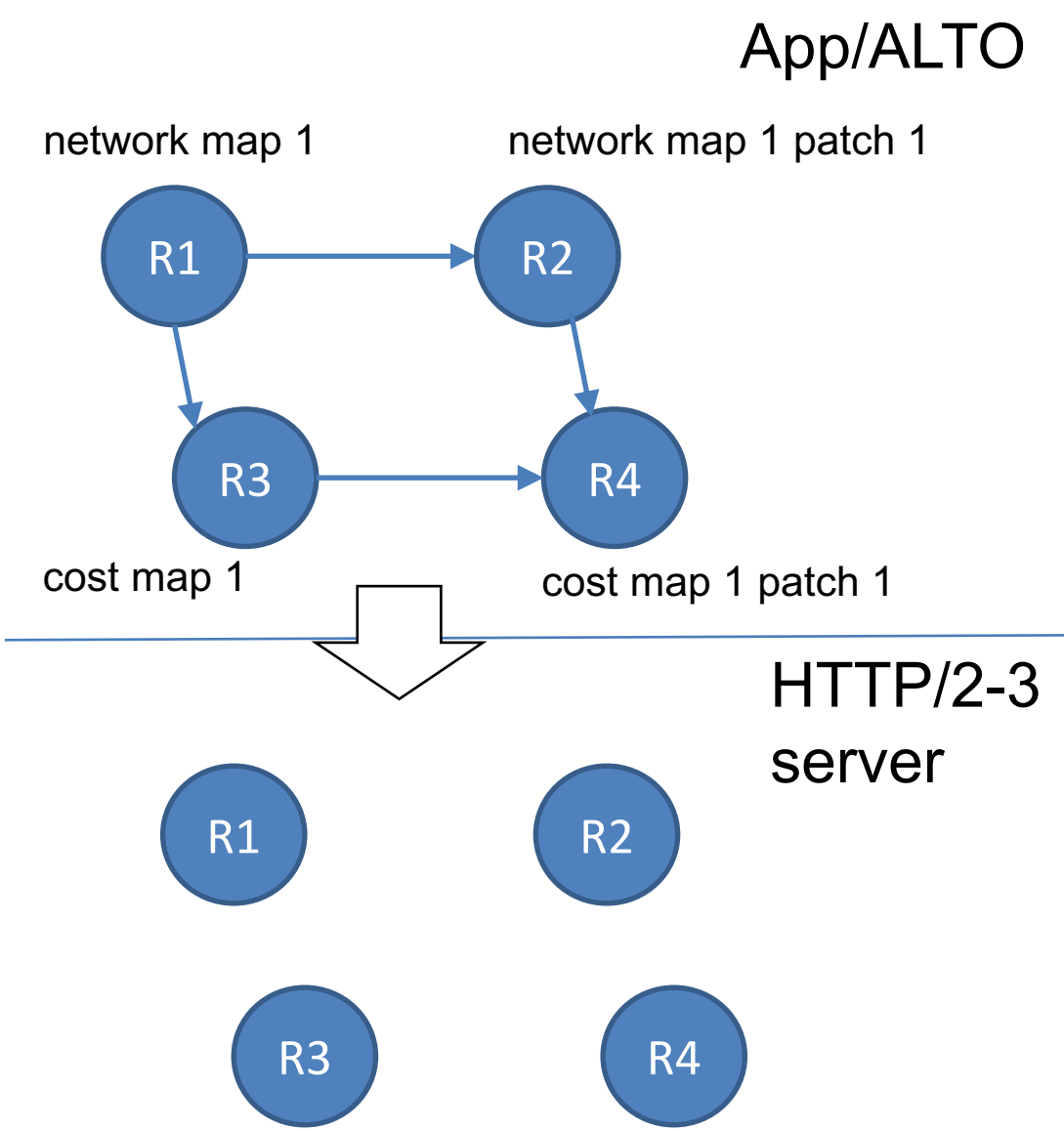
- ALTO specifies only: mapping each Ri to an independent HTTP/2-3 stream; HTTP does scheduling.
- Issue: HTTP could schedule R4, then R3, then R2 and then R1 in transmitting order

Design 2:

- ALTO specifies that server submits to the HTTP transport in DAG order: submit Ri only when what Ri depends on are finished: R1; R2/R3, R4
- Issue: Sliding window is large and transport can fit R1/R2/R/R4 into a single window

Design 3:

- ALTO indicates the dependencies to HTTP
- Issue: HTTP client can indicate a parent in req header, but this is signaling from client to server; what we need are (1) app signaling to HTTP server, (2) multiple dependencies



Discuss 3: How/Whether to Specify Settings

- Current draft allows client to specify two (HTTP/2) control knobs on server behaviors
  - 0x02 SETTINGS_ENABLE_PUSH (a BCP14 “MUST”)
  - 0x03 SETTINGS_MAX_CONCURRENT_STREAMS (a BCP14 “must”)
- HTTP/3 changed these settings (RFC9114) [SD review]
  - SETTINGS_ENABLE_PUSH (0x02):This is removed in favor of the MAX_PUSH_ID frame, which provides a more granular control over server push. Specifying a setting with the identifier 0x02 is HTTP/3 error.
  - SETTINGS_MAX_CONCURRENT_STREAMS (0x03):QUIC controls the largest open stream ID as part of its flow-control logic. Specifying it is HTTP/3 error
- Discussion: (1) remove them in the spec and discuss them in operations; (2) specify generic requirements statement

Discuss 4: Examples using HTTP/1.1 or Later

- Initial draft used HTTP/1.1; then switched to 1.1+2; current draft is only HTTP/2
- Style guide recommends using HTTP/1.1 but if we want to specify more control, we need a way to specify them

```
POST /tqs HTTP/2
Host: alto.example.com
Accept: application/alto-transport+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: TBD

{
    "resource-id": "my-routingcost-map"
}

HTTP/2 200 OK
Content-Type: application/alto-transport+json

{"mq": "/updates/streams/2718281828459"}
```

```
Server -> client PUSH_PROMISE in current stream:

PUSH_PROMISE
  - END_STREAM
    Promised Stream 4
    HEADER BLOCK
    :method = GET
    :scheme = https
    :path = /tqs/2718281828459/uq/101
    host = alto.example.com
    accept = application/alto-error+json,
                 application/alto-costmap+json

Server -> client content Stream 4:

HEADERS
  + END_STREAM
  + END_HEADERS
    :status = 200
    content-type = application/alto-costmap+json
    content-length = TBD

DATA
  + END_STREAM
 {
   "meta" : {
      "dependent-vtags" : [{
         "resource-id": "my-network-map",
         "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
```

Discuss 5: Media Type

- Finalize media type: a single media type (application/alto-transport+json) to encode all new data items

Next Step

- The authors will submit a new version as soon as the WG makes the decisions on Discuss 1-5 (in two weeks if possible)
  - Discuss 1.1 Allow client long pull
  - Discuss 1.2 Server push promise -> server put | + server put
  - Discuss 2: How much to specify ordering control: how much transport-awareness of app semantics
  - Discuss 3: How/whether to specify settings
  - Discuss 4: Examples using HTTP/1.1 or HTTP/2
  - Discuss 5: Media type comments

Backup Slides

Service	Type (Out if not labeled)	Core Information Structures	Main Size Var	Stability Expectation	Incremental Changes
Information Resource Directory [RFC7285]	application/alto-directory+json	Key-value store; Delegation	#resources	Stable	Add/delete resources
Network Map [RFC7285]	application/alto-networkmap+json	Key-value store: pid -> addrType -> array	#CIDRs	Stable	Add/delete CIDR from pid
Cost Map [RFC7285]	application/alto-costmap+json	Key-value store: srcPID -> dstPID -> value Depend on network map	#SRCPID * #DSTPID	More dynamic than network map	Update cost map entries
Filtered Map Services [RFC7285]	In: application/alto-networkmapfilter+json Out: application/alto-costmapfilter+json	In: selected srcPID, dstPID Key-value store: srcPID -> dstPID -> value Depend on network map	Filtered #SRCPID * #DSTPID	More dynamic than network map	Update cost map entries
Endpoint Property Service [RFC7285]	In: application/alto-endpointpropparams+json Out: application/alto-endpointprop+json	In: addr + prop Key-value store: addr -> prop -> value	#addr * #prop	Depend on property, can be dynamic or stable	Update property value
Endpoint Cost Service [RFC7285]	In: application/alto-endpointcostparams+json Out: application/alto-endpointcost+json	In: srcAddr x dstAddr Key-value store: srcAddr -> dstAddr -> value	#src * #dst	Can be more dynamic than cost map	Update cost value
Cost Calendar [RFC8896]	Extension to CostType	Calendar array	Previous * #num_intervals	Can be dynamic	Calendar window moves
Unified Property	In: application/alto-propmapparams+json; Out: application/alto-propmap+json	In: addr, prop Key-value store: addr -> prop -> value	#addr * #prop	Depend on property, can be dynamic or stable	Update property value
Path Vector	In: see costmap; Out: multipart/related;type=application/alto-costmap+json,	Cost map + unified property map	#src * #dst * #vec size	Depend on metric, can be dynamic or stable	Update path vector
CDNi Cap & Footprint	application/alto-cdni+json	Array; {capability-type: capability-value}	#footprint * #capability	Can be dynamic, bursty	Update capabilities

Performance and Effectiveness of Current Transport

Infrastructures	Basic Workload (ALTO SPEC)	Transport	Collecting metrics

• Benocs is fully open to use its infrastructure as an evaluation environment
• Greater Bay Network is also fully open to use its infrastructure as an evaluation environment
	1. (Filtered) Cost map: distribute inter-site performance metrics and calendar; routing changes + link dynamics => updated metrics
	2. Endpoint/unified property service: endpoint access status query/updates/bwe
	3. CDN node footprint & capability
	4. Flow direction (pointing to CDN nodes) using ECS
	5. Path vector providing available reservable bandwidth
		• HTTP/1.x per request full retrieval
		  • keep alive
		  • pipelining
		• HTTP/2, HTTP/3 per request, full retrieval
		• ALTO/SSE RFC8895 (on HTTP/1.x)
			• ALTO Server processing load
			• ALTO client processing load
			• Transport load (bytes)
			• Transport latency
			• Throughput
			• Scalability

General Space of Network->App Information Transport

Protocol/Base Reference	What information	How transported	Frequency
ALTO/RFC7285, RFC8895	Network map, cost map, unified property, …	HTTP/1.x client/server	Request/response; ALTO/SSE incremental push
ECN/RFC3168	Congestion notification	2 bits in IP Traffic Class header; ECN-Echo/CWR flags in TCP header	Per packet marking
NEF, SCEF<->AS/3GPP, …	Network capabilities/events -> AF	HTTP	Request/response
…			

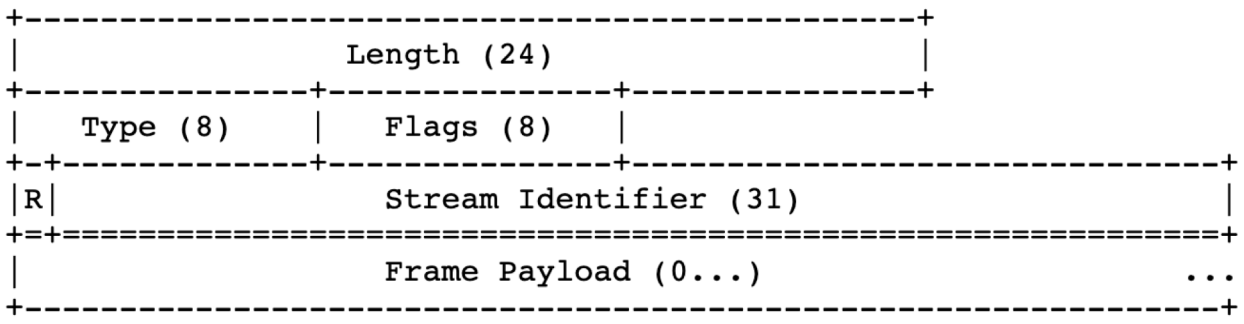


Figure 1: Frame Layout

The value 0x0 is reserved for frames that are associated with the connection as a whole as opposed to an individual stream.

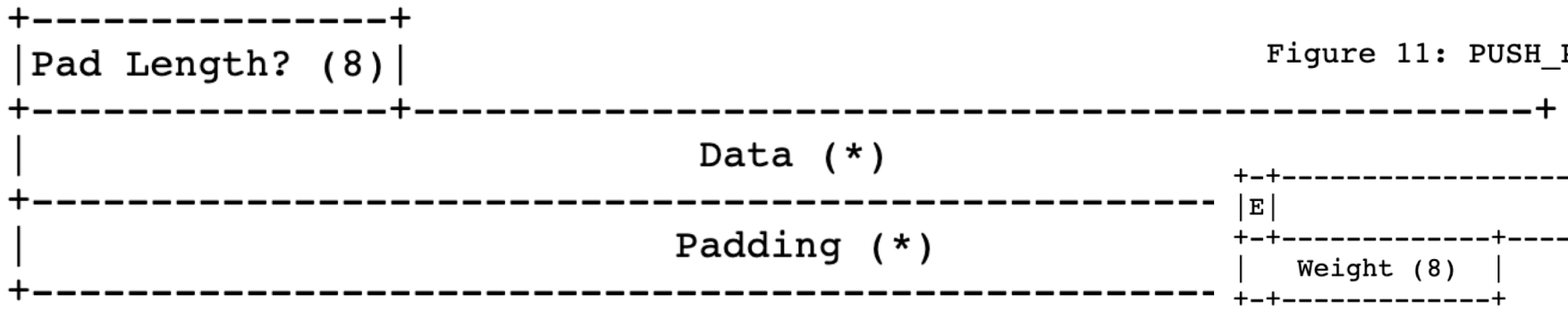


Figure 6: DATA Frame Payload

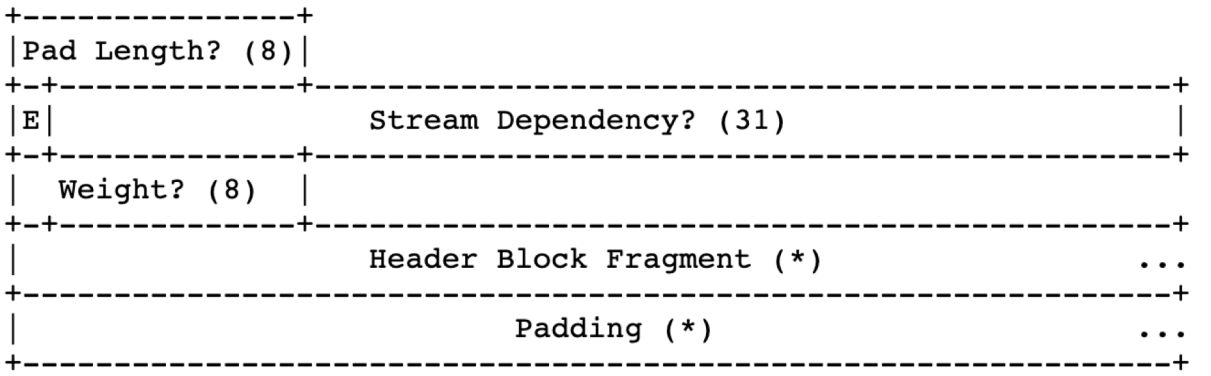


Figure 7: HEADERS Frame Payload

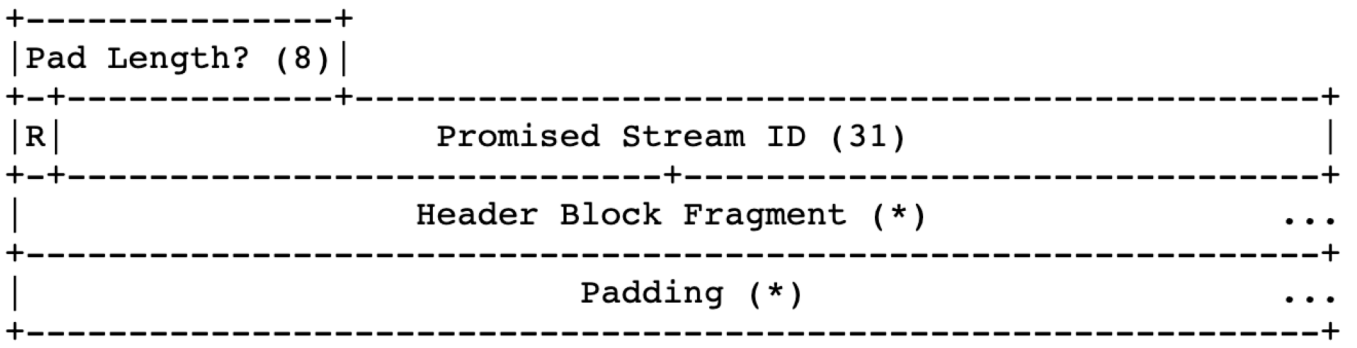


Figure 11: PUSH_PROMISE Payload Format

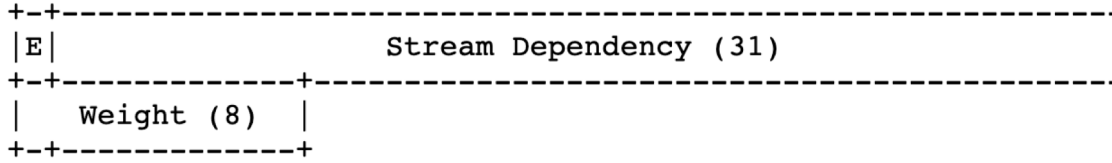
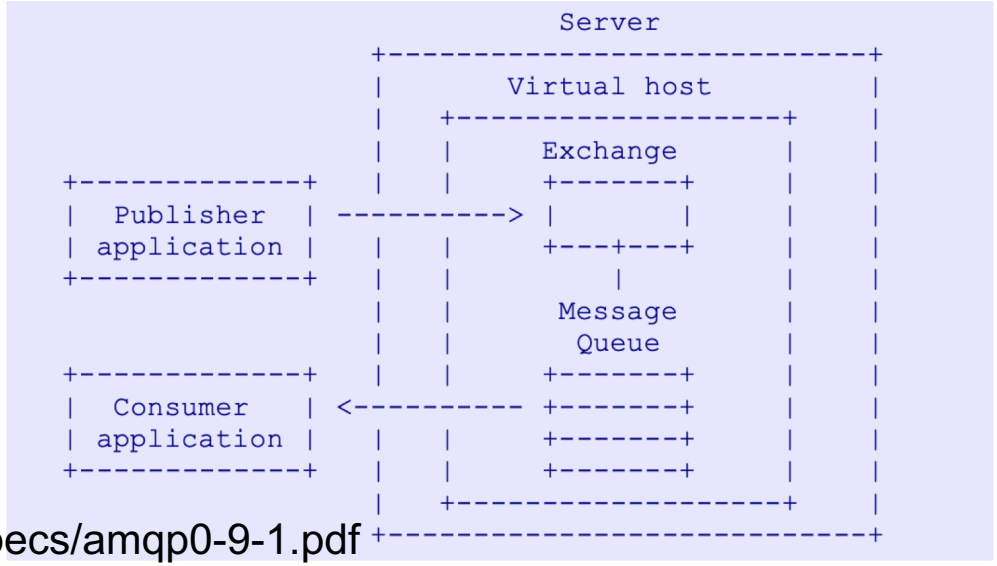


Figure 8: PRIORITY Frame Payload

Transport and Pub/sub

- What is missing
  - The design does not allow creation of generic message queues
  - Only the server can be the publisher
    - Clients publish info to be shared with other client
  - The design does not have the capability of Exchange (message router)
- Way forward: Keep simple
  - Broker for further discussion



https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf

Additional Information about Transport Queue

- Calendar semantics
  - Tell the client ALTO information (e.g., cost) for a future time point
  - Tell the client when the next information will be released, it is the time that the info is released is distributed, not the value [support]

Negotiation

- Task: to fully specify the complete set of negotiation parameters
  - Use HTTP/2 (default), HTTP/3
  - Incremental updates queue encoding
    - Chosen by server, but must be specified as HTTP Accept header of client
      - Initial connection setup: client sends list of incr update mime types, and server returns the subset which it can use (Vary header in response)
  - Concurrency level (e.g., Rucio controller need to monitor info for a large number of clients) stream control