MASQUE
Using QUIC Datagrams with HTTP/3

draft-ietf-masque-h3-datagram

IETF MASQUE Interim – Virtual – 2021-04

David Schinazi – dschinazi@google.com
Lucas Pardue – lucaspardue.24.7@gmail.com

Let's start from the basics

QUIC DATAGRAM frame just has a Payload

HTTP/3 is a multiplexed protocol

If we have multiple CONNECT-UDP and CONNECT-IP requests on the same connection, we need a way to associate each datagram with the right request

Solution: have the DATAGRAM frame Payload start with a varint

```
HTTP/3 DATAGRAM Frame {
  Flow Identifier (i),
  HTTP/3 Datagram Payload
(..),
}
```

But wait, there's more! How do you associate a varint with a request?

Some requirements

False start: client may send a datagram before receiving the HTTP response

Intermediary support

Extensibility of CONNECT-UDP/IP (without changing intermediaries)

Example: CONNECT-UDP without extensions

Every datagram payload maps to a UDP payload

You just need to associate the datagram varint with the request

```
HTTP/3 Datagram Payload = {
  UDP Payload (..),
}
```

Example: CONNECT-UDP + timestamp extension

Every datagram payload maps to a timestamp followed by UDP payload

```
HTTP/3 Datagram Payload = {
  Timestamp (64),
  UDP Payload (..),
}
```

When receiving a datagram, how do you know whether to parse with this format or with the original extension-less CONNECT-UDP format?

Example: CONNECT-IP + compression extension

Start off with all traffic sent as: Datagram = IP Packet (uncompressed)

Realize that a 5-tuple is a lot of traffic, negotiate compression for that 5-tuple

Packets for that 5-tuple can now be sent as: Datagram = compressed format

Multiple formats coexist on a single request

Not possible to use separate HTTP requests here because different requests can be routed to different backends

Updated requirements

False start: client may send a datagram before receiving the HTTP response

Intermediary support

Extensibility of CONNECT-UDP/IP (without changing intermediaries)

Multiple datagram payload formats multiplexed in a single request

Negotiation of new datagram payload formats mid-stream

And now for the design discussion!

How to associate datagrams with requests and in-request context (e.g. compressed vs not)

Single-layer design (currently in draft)

Datagram payload contains one varint: flow ID

Flow ID is connection-wide, per-hop, maps to request and additional in-request context

Downside: intermediaries need to be involved in the in-request negotiation

Two-layer design

Datagram payload contains two varints: stream ID and flow ID

Stream ID is connection-wide, per-hop, maps to request

Flow ID is per-request, end-to-end, maps to additional in-request context (e.g. compressed vs uncompressed)

Downside: uses one more varint on the wire

Are flow IDs unidirectional or bidirectional? TODO

Bidirectional design (currently in draft)	**Unidirectional design**
One single namespace, even=client-initiated, odd=server-initiated	Two namespaces, one per endpoint
Inherently negotiated, peer echoes the registration to indicate acceptance	Unilateral declaration, no negotiation
	Downside: DoS vector by asking peer to save infinite state
	Downside: some extensions (e.g. QUIC-aware CONNECT-UDP) want the ability to reject registrations

How to negotiate flow IDs

HTTP header design (currently in draft)

Creates mapping at start of stream

```
Datagram-Flow-Id = 2, 4; foo=bar
```

Downside: in the one-layer design, it's effectively a hop-by-hop header which is frowned upon in HTTP/3
Downside: it can't create an association mid-stream
Downside: the header format as a list with parameters didn't receive much love

"REGISTER_FLOW_ID message" design

Can create mapping mid-stream

Message carries the flow ID and a way to convey format/semantics/extension-data

Can be implemented via an HTTP/3 frame (or other options, but let's not bikeshed right away)

(Assuming two-layer design)
Is the flow ID layer optional?

Some HTTP extensions might not need flow IDs, and want to save one byte

Proposal (assuming REGISTER_FLOW_ID message):
add separate REGISTER_SINGLE_FLOW message which indicates that no flow IDs are in use

REGISTER_FLOW_ID Message Design

Flow ID zero

Reserve flow ID 0 as a control channel

Means control messages can be sent unreliably, requires a RELIABLE_DATAGRAM HTTP/3 frame

Intermediaries don't need to know that REGISTER_FLOW_ID exists

Separate frame

New HTTP/3 REGISTER_FLOW_ID frame

Cleaner separation

Intermediaries need to understand the REGISTER_FLOW_ID frame in addition to understanding DATAGRAM (and potentially RELIABLE_DATAGRAM)

REGISTER_FLOW_ID Message Design, part 2

No abstraction

The message (flow ID 0 | frame) means REGISTER_FLOW_ID

Slightly less code

One layer of abstraction

The message (flow ID 0 | frame) means "control message" which starts with a varint Message Code, and code 0 = REGISTER_FLOW_ID

Requires IANA registry

Too much extensibility is reminiscent of SNI, an extensibility joint that rusted shut

This extensibility would allow new messages to be defined end-to-end without modifying intermediaries

REGISTER_FLOW_ID Message Design, part 3

The register message carries a flow ID, and associated information

Needed to know the format/semantics/extension-data of this flow ID

Examples:

- Regular CONNECT-UDP
- CONNECT-UDP with timestamp
- CONNECT-IP but with compressed IP/port of 192.0.2.33:443

How do we encode this?

Proposal: list of text key-value pairs, e.g. `"ip=192.0.2.42,port=443"`

RELIABLE_DATAGRAM frame

Required in the flow ID 0 control channel design

Nice to have in other designs

If not present, require a way to send datagrams reliably on the DATA stream

Requires intermediary support, which adds work now but also means that adding this later might be harder

So, do we need this?

MASQUE
Using QUIC Datagrams with HTTP/3

draft-ietf-masque-h3-datagram

IETF MASQUE Interim – Virtual – 2021-04

David Schinazi – dschinazi@google.com
Lucas Pardue – lucaspardue.24.7@gmail.com