MASQUE
Using QUIC Datagrams with HTTP/3

draft-ietf-masque-h3-datagram

IETF 110 – Virtual Prague – 2021-03

Lucas Pardue – lucaspardue.24.7@gmail.com

The 15-second summary

draft-ietf-masque-h3-datagram adopted

HTTP/3 DATAGRAM is like QUIC DATAGRAM, but the frame payload must start with a Flow ID!

Datagram flows are logical bidirectional exchanges of HTTP/3 DATAGRAMS. Supports demultiplexing.

Use of flows benefits from peer coordination i.e. signalling what a flow ID maps to. So associate to HTTP messages.

Design premise 1: Flow ID

All flows have an identifier, value is chosen by a flow identifier allocation service

Flow ID is a 62-bit integer (0 to 2^62-1)

When carried in frame, encoded as a QUIC variable-length integer

When carried in header, encoded as an sf-int (0 to 10^15-1)

Design premise 2: H3_DATAGRAM setting

Some HTTP/3 applications might not want the demultiplexing afforded by flow ID

- Might want other rules for using QUIC DATAGRAM frame payload when HTTP/3 is negotiated

H3_DATAGRAM setting allows endpoints to negotiate the use of flow IDs defined in this specification

Design decision: associating flows with streams

Reuse the HTTP request stream ID

Simpler

Forces 1-to-1 mapping from request to ID

Only uses 25% of available IDs
    Leads to longer varint encoding

Datagram-Flow-Id HTTP Header

Distinct namespace

Requires explicitly negotiating via header

Allows many-to-1 mapping from request to ID
    Used by draft-pauly-masque-quic-proxy

Allows potentially reusing flow IDs

More extensible

We need to pick one, other drafts depend on us.

Multiplicity

1. 1 stream - 1 flow
    a. Minimum: all implementations of HTTP/3 DATAGRAM must support this
2. Many streams - 1 flow
    a. Reference a flow established by one HTTP message from another, see draft-pauly-masque-quic-proxy
3. 1 stream - many flows
    a. Establish many flows via a single HTTP message.
    b. One mandatory flow for the standard behaviour, can always rely on it
    c. Additional flows enhance the behaviour of the mandatory flow
    d. Additional flows might be extensions that are not supported by the peer.
        i. Datagram-flow-id echoing allows a client to detect if a server does not support an extension.

Supporting multiplicity with sf-list

Datagram-Flow-Id = sf-list supports all multiplicity requirements

In order to support ignoring additional flow IDs for unknown extensions, spec defines the notion of names, encoded as the first parameter of each list member. The mandatory flow is unnamed.

```
Datagram-Flow-Id = 2, 4; never-gonna-give-you-up
```

Under the hood, the “name parameter” is a true sf-bool, whose encoding rules make it look nice and concise. Imagine it more like

```
Datagram-Flow-Id = 2, 4; never-gonna-give-you-up=1
```

Issue #24 – parameters should be not be names

Use parameters more conventionally and clarify if spec's current design allows each item to have more parameters than a name.

Suggestion 1:

```
Datagram-Flow-Id = 2, 4; never=give-you-up,
                      6; never=let-you-down
```

Suggestion 2:

```
Datagram-Flow-Id = 2, 4; name=never-give-you-up,
                      6; name=never-let-you-down
```

Another alternative ...

Issue #24 - alternate encoding of Datagram-Flow-Id

Use regular parameters, and reserve a "default" parameter

```
Datagram-Flow-Id = 42; ecn=not-ect default, 44; ecn=ect0,
                   46; ecn=ect1, 48; ecn=ce
```

- If the proxy doesn't support the ECN extension, it will ignore that parameter and only use flow ID 42 which is default
- Allows composition of extensions

```
Datagram-Flow-Id = 42; evilbit=evil never=give-you-up default,
                   44; evilbit=nice never=give-you-up,
                   46; evilbit=evil never=let-you-down,
                   48; evilbit=nice never=let-you-down
```

Issue #24 - pick an option

Can we agree to keep Datagram-Flow-Id, and can we pick a format option?

1. `Datagram-Flow-Id = 2, 4; never=give-you-up`

2. `Datagram-Flow-Id = 2, 4; name=never-give-you-up`

3. ```
   Datagram-Flow-Id = 2; evilbit=evil never=give-you-up default
                      4; evilbit=nice never=give-you-up
   ```
   ```

Issue #22 – Reusing flow IDs is racy

Flow ID is carried in every frame, using small IDs can be attractive

PR #20 introduced the notion that flow allocation service supports “retire” and “reuse” of flow IDs.

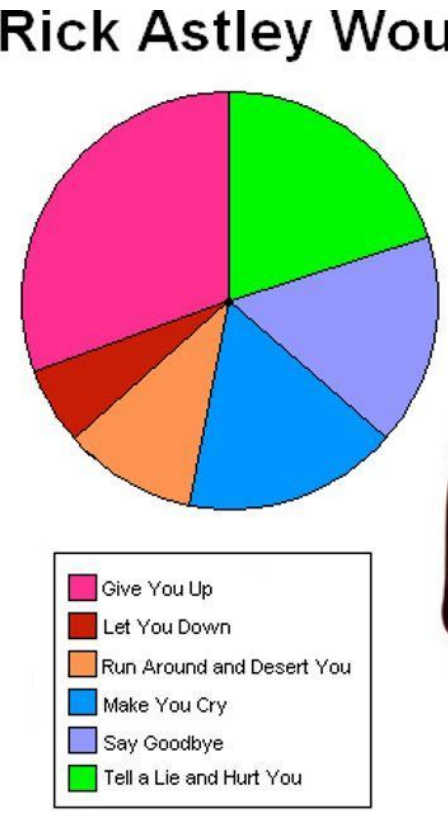
Application needs to tell the flow allocation service when to retire

Detecting when it is suitable or safe to retire might be hard depending on implementation. Risk that recycled flow IDs are used incorrectly.

We have a 62-bit integer, do we need the potential complexity for reuse? Is anyone opposed to dropping this feature?

MASQUE

HTTP/3 DATAGRAM



draft-ietf-masque-h3-datagram

IETF 110 – Virtual Prague – 2021-03

Lucas Pardue – lucaspardue.24.7@gmail.com