MASQUE CONNECT-UDP

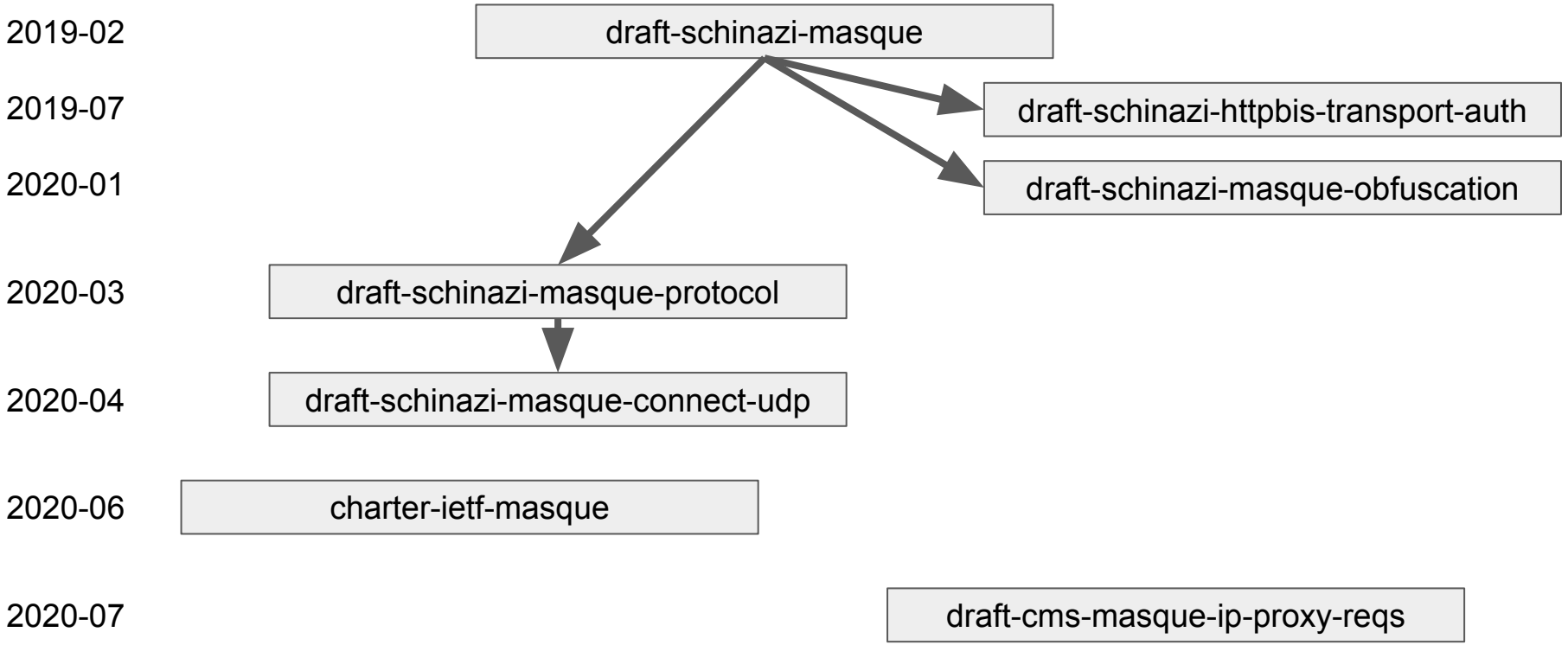
draft-schinazi-masque-connect-udp

IETF 108 – Virtual – 2020-07

David Schinazi – dschinazi@google.com

Some Historical Context

MASQUE: Multiplexed Application Substrate over QUIC Encryption



HTTP CONNECT

RFC 2817 from 2000

Supported in all versions of HTTP (since 1.1)

Clients instructs Proxy to open a TCP connection to host:port,
and forward stream data in both directions

Data is sent in the bidirectional stream that carried the CONNECT request

TCP is great, but what about UDP?

We need something similar for UDP, to carry WebRTC, QUIC, DTLS, etc.

Reusing CONNECT is not possible, because if the proxy does not support this new mode, we do not want it fall back to TCP on the proxy–server segment

HTTP CONNECT-UDP

Can be supported in all versions of HTTP (since 1.1)
(Needed for networks that block UDP, and for 1.1-only intermediaries)

Clients instructs Proxy to open a UDP connection to host:port,
and forward datagram data in both directions

Data is sent in the bidirectional stream that carried the CONNECT request

```
CONNECT-UDP server.example.com:443 HTTP/1.1
Host: server.example.com:443
```

Optimization for HTTP/3: use QUIC DATAGRAM frames instead of streams

Using QUIC DATAGRAM frames from HTTP/3

Currently relying on draft-schinazi-quic-h3-datagram

When QUIC is in use and ALPN=h3,

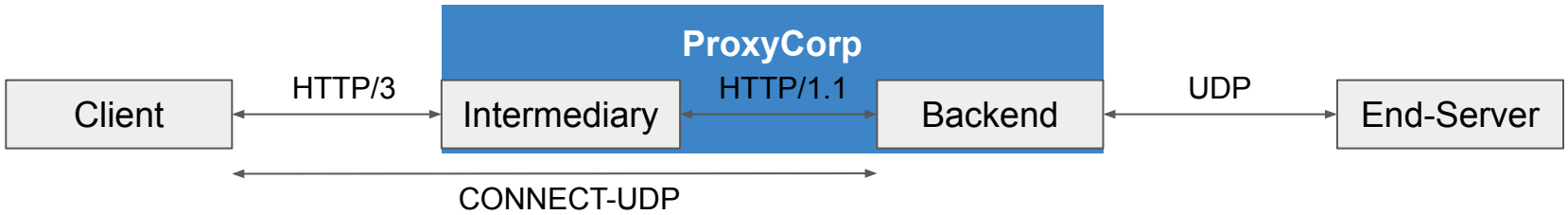
- Every QUIC DATAGRAM frame starts with a Flow Identifier (62-bit integer)
- Both endpoints provide a flow allocation service to get unique identifiers
- The protocol to negotiate these flow IDs is not defined in that draft

CONNECT-UDP carries the new "Datagram-Flow-Id" header to indicate flow ID

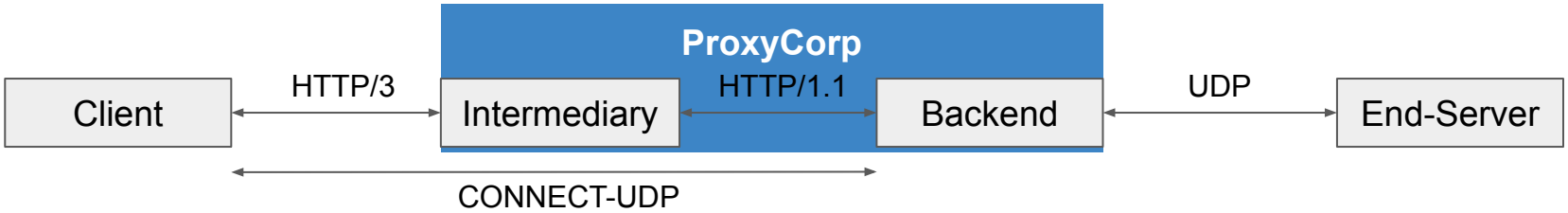
```
:method = CONNECT-UDP
:authority = server.example.com:443
Datagram-Flow-Id = 42
```

Chaining Multiple HTTP Proxies

Even though ProxyCorp appears to be a single machine to the client,
it can be implemented as one or more HTTP intermediaries leading to a backend



Chaining Multiple HTTP Proxies



Chaining is straightforward when sending UDP payloads in the request stream

Negotiating Datagram Flow ID across multiple proxy hops is non-trivial,
as flow IDs are a property of the transport, and aren't end-to-end

Should we make "Datagram-Flow-Id" hop-by-hop,
and send the "Connection" header listing it to ensure it isn't forwarded?

Out of Scope – Potential Extensions

CONNECT-UDP aims for simplicity, goal is to produce a minimum viable product that allows proxying UDP-based protocols over HTTP

The following topics are considered future work left for extensions:

- ICMP
- UDP Checksum
- DSCP
- ECN
- IPv6 Flow Label
- NAT Traversal (TURN, etc.)
- Fragmentation
- IP Options
- UDP Flags
- MTU Discovery
- Nested Congestion Control

Should we Merge these Drafts?

draft-schinazi-masque-connect-udp
draft-schinazi-quic-h3-datagram

Is there a need for HTTP/3 datagrams that are not related to a request stream?

If not, then it might be best to define the protocol to negotiate flow IDs in the same document as where flow IDs are defined

A Pattern is Emerging...

CONNECT-UDP, CONNECT-QUIC, CONNECT-IP, WebTransport, ...

All of these have similar properties:

- Negotiation is performed via an HTTP request that resembles CONNECT
- Negotiation of a DATAGRAM Flow ID to allow multiplexing

Should we unify all of these on a single new method?

- CONNECTX (insert naming bikeshed here)
- Carries a "Connectx-Protocol" header that MUST be present
- The protocol determines server behavior, MUST reject unknown protocols
- Multiple protocols can reuse the "Datagram-Flow-Id" header

Next Steps

Answer questions from previous slides then ask for Working Group adoption?

MASQUE CONNECT-UDP

draft-schinazi-masque-connect-udp

IETF 108 – Virtual – 2020-07

David Schinazi – dschinazi@google.com