MASQUE
HTTP Datagrams and CONNECT-UDP

draft-ietf-masque-h3-dgram
draft-ietf-masque-connect-udp

IETF 111 – Virtual – 2021-07-26

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

draft-ietf-masque-(h3-dgram|connect-udp) – IETF 111 – Virtual – 2021-07-26
Previously, on MASQUE...

We are building CONNECT-UDP, like CONNECT but for UDP!
We want it to work over all versions of HTTP, and across intermediaries
When over HTTP/3, we want to leverage the QUIC DATAGRAM frame

There is interest in datagrams beyond CONNECT-UDP, so we split the draft into HTTP Datagrams + CONNECT-UDP

We had an interim in 2021-04, focused on the design of HTTP Datagrams

We redesigned everything, and after discussion on the list, merged some PRs
Interlude: interop results

<table>
<thead>
<tr>
<th>server →</th>
<th>quiche</th>
<th>Apple QUIC</th>
<th>Google</th>
<th>Ericsson</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quiche</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apple QUIC</td>
<td></td>
<td></td>
<td>QHC DENRO</td>
<td>QHC DENRO</td>
</tr>
<tr>
<td>Google</td>
<td>-</td>
<td></td>
<td>QHC DENRO</td>
<td>QHC DENRO</td>
</tr>
<tr>
<td>Ericsson</td>
<td>-</td>
<td></td>
<td>QHC DENRO</td>
<td>QHC DENRO</td>
</tr>
</tbody>
</table>

Also: iCloud Private Relay means Apple and CDN partners have CONNECT-UDP deployed in production!

What do HTTP Datagrams look like now?

HTTP Datagram {
    Quarter Stream ID (i),
    [Context ID (i)],
    HTTP Datagram Payload (..),
}

HTTP Datagrams are now strongly associated with HTTP request streams

They can optionally carry a Context ID which allows multiplexing multiple flows of datagrams for the same request (e.g. compression, priorities)

Since you can have different formats or encodings, those need to be communicated to the peer
Stream Chunks went away

In previous drafts, CONNECT-UDP would use a sequence of "Stream Chunks" in the DATA stream – sequence of TLVs

But intermediaries need to convert from QUIC DATAGRAM to Stream Chunk

Since this broke the architectural purity of intermediaries not changing the DATA stream, we went with a different approach...
Capsules!

In the presence of HTTP intermediaries, HTTP/2 and HTTP/3 frames are hop-by-hop, not end-to-end.

How do we get information across end-to-end if the DATA stream isn't suitable?

Capsules are a new HTTP/2 and HTTP/3 frame that is forwarded by intermediaries, even if the intermediary doesn't know the capsule type.

This allows defining new capsules without having to modify intermediaries.

For HTTP/1, we'll define a new protocol that is a sequence of capsules.
We should make sure we have consensus on this

Are folks OK with the change from Stream Chunks to Capsules?

Both would work

One main difference though: capsules mean the DATA stream is still available

Both require modifying intermediaries the first time around

Neither plays nicely with existing HTTP semantics

But the architecture implications are different
Of datagrams and capsules

DATAGRAM capsule carries an HTTP datagram

Allows sending HTTP Datagrams over HTTP/2 (and HTTP/1)
Contexts and registrations

Contexts are end-to-end, they are opaque to intermediaries.

Each request has its own namespace of context IDs – 62bit integers.

Even ones are client-initiated, odd ones are server-initiated.

Both can be used bidirectionally.

"Registration" is the act of unilaterally declaring the semantics of a context, uses REGISTER_DATAGRAM_(NO_)CONTEXT capsule.

Context IDs can be closed by either endpoint, uses CLOSE_DATAGRAM_CONTEXT capsule.
Example of a context extension

Client                                             Server

STREAM(44): CAPSULE -------->
    Capsule Type = REGISTER_DATAGRAM_CONTEXT
    Context ID = 0
    Context Extension = {}

    /* Endpoint happily exchange encapsulated IP packets */
    /* using Quarter Stream ID 11 and Context ID 0. */
    /* After performing some analysis on traffic patterns, */
    /* the client decides it wants to compress a 5-tuple. */

STREAM(44): CAPSULE -------->
    Capsule Type = REGISTER_DATAGRAM_CONTEXT
    Context ID = 2
    Context Extension = {IP_COMPRESSION=tcp,192.0.2.6:9876,192.0.2.7:443}
CONNECT-UDP

Draft was on hold while we were redesigning HTTP Datagrams

Now that HTTP Datagrams support all versions of HTTP, CONNECT-UDP is much simpler

Now reduces to: send a CONNECT-UDP request, then encode UDP payloads as HTTP Datagrams
Now everyone's favorite: open GitHub issues!
#66: Context ID extensibility

Register context capsules carry a Context Extensions field

Meant to allow conveying semantics of a context

It's a sequence of TLVs where types are registered in an IANA registry (designed to resemble QUIC transport parameters)

This is maximal and can encode pretty much any extension under the sun, but is it what we want?

What about extension composability?
If extension A adds a timestamp and B adds an ECN bit, do we want A+B?
#65: HTTP Datagrams and MTU

If an application tries to send an HTTP Datagram payload that won't fit inside a QUIC packet on this path, it can either get dropped or be sent as a capsule.

Current draft tells intermediaries to avoid converting from datagram to capsule as that prevents end-to-end PMUTD.

We could build a protocol feature that communicates the MTU.

Or this could make for a good extension to HTTP Datagrams.
#23: CONNECT-UDP scheme

CONNECT-UDP is not CONNECT so it must carry the :scheme pseudo-header.

The value really does not matter.

That makes it the perfect bikeshed.

Can we use https and move on?
CONNECT-UDP or Extended CONNECT

CONNECT-UDP currently defines its own method

WebTransport currently uses Extended CONNECT (RFC 8441)

Since the two are pretty similar, should they make the same choice here?

Disclaimer: though many WebTransport enthusiasts are here today, anything we agree on in MASQUE would need to be discussed in WEBTRANS on Friday, and on both mailing lists.
MASQUE
HTTP Datagrams
and CONNECT-UDP

draft-ietf-masque-h3-datagram
draft-ietf-masque-connect-udp

IETF 111 – Virtual – 2021-07-26

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