

Retransmission of HTTP/3 Datagrams

[draft-yang-masque-dgram-retrans-01](#)

Furong Yang, **Yanmei Liu**, **Yunfei Ma**

yfr256538@alibaba-inc.com

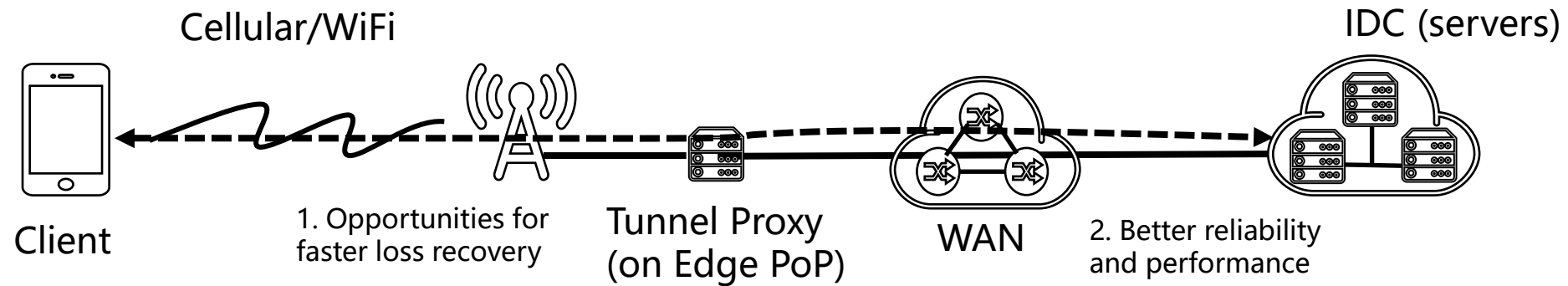
miaoji.lym@alibaba-inc.com

yunfei.ma@alibaba-inc.com

MASQUE IETF 116, March 2023, Yokohama

Background

- Our use-case:
accelerating QUIC in Mobile App via UDP tunneling



Problems

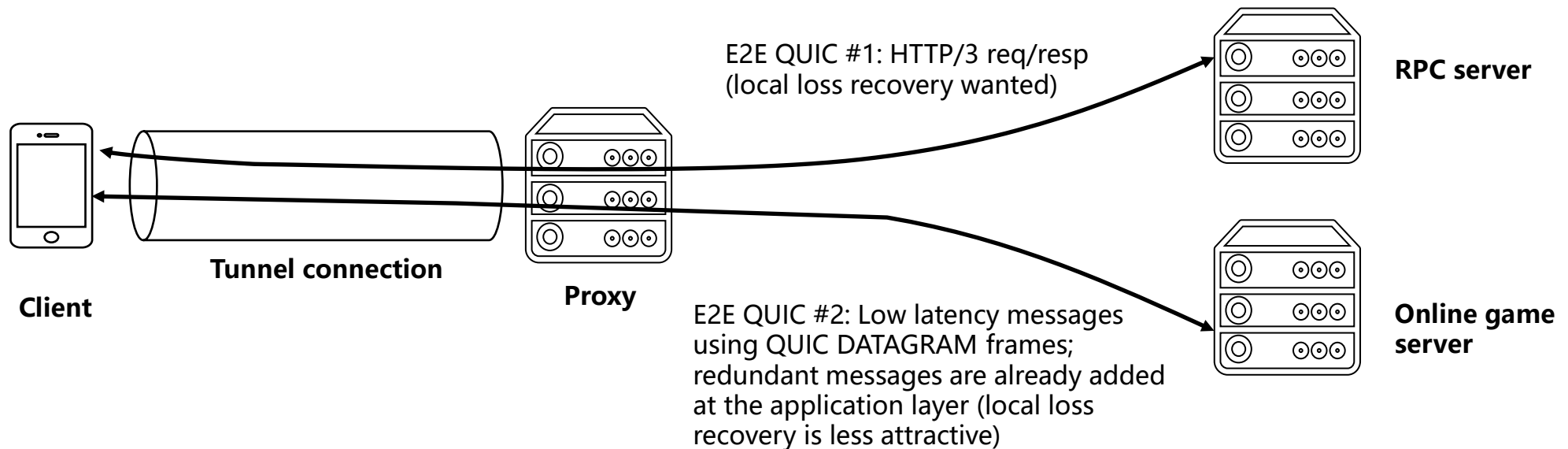
- Two ways of tunneling QUIC connections in MASQUE
 - Reliable
 - E2E QUIC packets -> HTTP/3 Datagrams -> DATAGRAM Capsule (QUIC stream)
 - Unreliable
 - E2E QUIC packets -> HTTP/3 Datagrams -> QUIC DATAGRAM frame
- Pros and cons
 - Reliable
 - local loss recovery between the client and the proxy
 - HoL-blocking (parallel E2E QUIC streams are serialized into one QUIC stream)
 - Unreliable
 - no HoL-blocking but no local loss recovery as well
- Retransmission of datagrams is appealing for performance enhancement (optional, but useful when packets are lost on the last mile)

Which layer to retransmit DGRAMs ?

- Option1
 - Add a transport parameter at the QUIC transport layer to negotiate whether QUIC should retransmit DATAGRAM frames
 - Connection-level, inflexible (not all tunneled connections need this feature)
- Option2
 - Configure the client and proxy to retransmit HTTP/3 Datagrams when they are declared lost by the transport layer
 - But how?

Basic design principles

- Each tunnel should have its own configuration (flexibility)



- We should be able to limit retransmission overhead
 - Lost packets could also be retransmitted by the E2E connection

Extension for HTTP/3 DGRAMs retrans.

- Adding a new boolean-valued Item Structured Field “DG-Retrans: ?1” to negotiate the use of this extension
- Using “Retransmission Limit” to control how many times an HTTP/3 DGRAM can be retransmitted
 - For client->server packets, the client unilaterally decides this limit
 - For server->client packets, the client uses a new Capsule Type SET_H3_DGRAM_RETX_LIMIT to inform the proxy this limit

Client

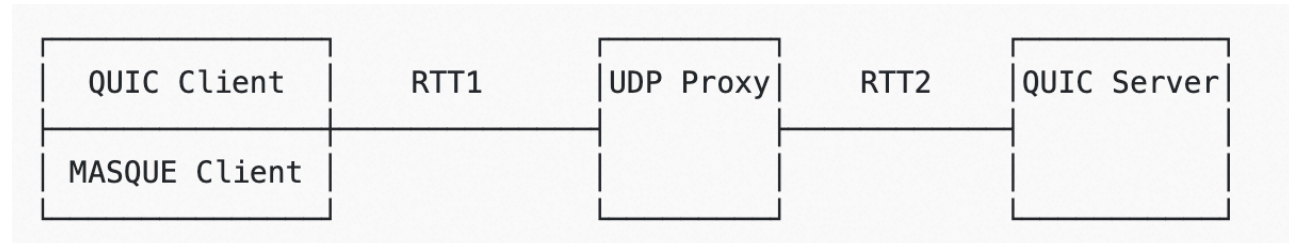
Proxy

```
STREAM(32): HEADERS
:method = CONNECT
:protocol = connect-udp
:scheme = https
:path = /target.example.com/443/
:authority = proxy.example.org
DG-Retrans = ?1
capsule-protocol = ?1
```

```
STREAM(32): DATA
SET_H3_DGRAM_RETX_LIMIT {
  Capsule Type (i) = 0xba..0xbb,
  Capsule Length (i),
  [Context ID (i)],
  Retransmission Limit (i),
}
```

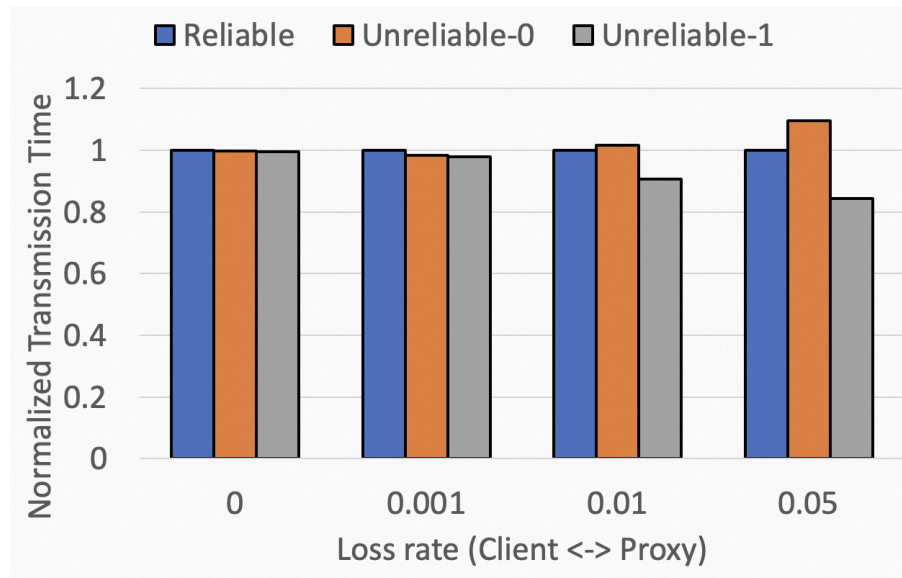
Why Retrans. Limit? How to set it?

- Why?
 - We want to have some control over the retransmission overhead, but it may not be the best way
 - As @Ben Schwartz suggested, the client could also report some transport performance metrics to the proxy using Capsules, which may make the proxy more adaptive on its retransmission policy
- How to set it?
 - A possible way in our use-case: $RTT2/RTT1$
 - The idea is to stop retransmissions once the E2E connection would have done it



Preliminary experimental results

- Periodical concurrent HTTP/3 req/resp
 - Client <-> Server: 20 parallel 5KB requests every 500ms
 - Client <-> Proxy: RTT 30ms, BW 100Mbps, congestion control BBR
 - Proxy <-> Server: RTT 32ms, BW 100Mbps, congestion control BBR
 - Random losses are added on Client <-> Proxy



1. Compared with reliable mode, unreliable-1 achieves better performance due to mitigating HoL blocking issues

2. Compared with the vanilla unreliable mode (unreliable-0), unreliable-1 (Retrans. Limit = 1) achieves better performance due to local loss recovery

Open discussion

- Is there a better way to replace Retrans. Limit?
 - Retrans. Limit serves as the upper bound (to control the overhead)
 - The proxy/client collects some network measurements to adaptively decide if a lost HTTP/3 Datagram should be retransmitted
 - What network measurements?
 - For instance, RTTs, estimated BW of tunnel and E2E connections?
 - loss rates?
 - How should the adaptation logic be done?

Open discussion

- Impact on E2E congestion control (CC)
 - For non-loss-based CC (e.g. BBR), it seems fine
 - For loss-based CC
 - Hiding losses by local retransmissions may delay loss-based E2E congestion controllers to react to congestion, leading to bloated sending buffers in the tunnel connection
 - Some AQM mechanisms (e.g. RED) could be introduced in the tunnel connection to actively drop queued packets to alleviate this problem

Open discussion

- Cascaded proxies
 - In our use-case, the local loss recovery is only desired for the Client <-> Proxy (first hop) link
 - Should we forward the Capsule(`SET_H3_DGRAM_RETX_LIMIT`) to the next hop (if there is one)?

