Multicast Extensions for QUIC

IETF 118, MBONED

draft-jholland-quic-multicast

Max Franke
Louis Navarre

Outline

- Quick draft recap
- Developments since 114
- Current state of the draft
- Reference implementation(s)
- Next steps

Basic idea

- Use QUIC unicast connections as anchor for multicast
  - Client declares multicast support and limits (e.g. max rate)
  - Server picks fitting SSM channel(s) and tells client to join them
  - Client attempts join and receives data packets over multicast
- Unicast connection is used for integrity/crypto keys
  - Similar to AMBI
  - Each packet sent over multicast has integrity frame (i.e. checksum) associated with it
  - Multicast packets are encrypted
- Client ACKs each packet over unicast
  - Server can choose to retransmit lost packets over unicast or (if many clients lost it) over multicast again
  - Guarantees reliability
- Multicast only works from server to client
- Multicast packets are also part of the QUIC connection, client does not need to differentiate

Problems this approach solves

- Feasible way to get multicast into browsers via QUIC
- Encryption for packets (though weak as all receivers can decrypt)
- Integrity prevents injection of packets by attackers
- Reliability means application layer does not have to worry about FEC etc.
- Since it is still just a QUIC connection to the application, support only requires enabling a flag
- Pushes large data packets to multicast while only keeping integrity frames over unicast, though those can potentially be pushed via multicast as well

Current state of the draft

- Version -03, making continuous progress on fleshing out the draft further
- Added FEC to reduce number of retransmissions
  - https://datatracker.ietf.org/doc/draft-michel-quic-fec/
- Multipath QUIC support
- Two decisions that we are debating:
  - Integrity - Frames vs. signatures
  - Reliability - ACK vs. NACK

Forward Erasure Correction recovery on the multicast channel

- Could rely on draft-michel-quic-fec-01
  - https://datatracker.ietf.org/doc/draft-michel-quic-fec/
- Each (e.g.) STREAM frame is encapsulated in a source symbol
- The source can generate REPAIR frames on the multicast channel
  - Proactively
  - Based on feedback from the clients
- A single REPAIR frame can recover different losses on distinct clients
- RFC9265
  - Do not hide losses to the congestion control
  - Only send REPAIR frames if allowed by the congestion control algorithm

Multipath QUIC to construct the multicast channel

- draft-ietf-quic-multipath to be standardized soon (hopefully)
  - https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/
- For the client, the multicast channel can be viewed as a second path using a different key material
- This enables seamless transition between unicast and multicast for the client

- This requires change from the current version of draft-ietf-quic-multipath
  - Need a different crypto material for each path

What to do with integrity?

- Two possible approaches as mentioned:
  - AMBI style integrity frames
  - Signatures that authenticate per packet or per stream
- Both approaches have advantages/disadvantages in different scenarios
- Do we include just one and if yes which one?
- Do we include both as options and let them be negotiated?

Feedback on/or experiences on this more than welcome.

Reference implementation(s)

- Gave up on Chromium implementation
- Somewhat working python (aioquic) open source implementation
  - Missing some parts, but all the frames, transport parameters etc. are there, sending and receiving of multicast packets works
  - Integrity and Acks not implemented (so far)
- Also existing quiche (cloudflare) implementation
  - Does FEC and NACKs
  - No flow of congestion control mechanics (i.e. channel limits etc.)

Next steps

- Make decisions on open issues (ACK and Integrity)
  - Potentially offer choice between AMBI and ALTA style integrity?
  - → Push AMBI drafts forward
- Move on with implementations, potentially compare the two
- Present new version in QUIC-WG (potentially at 119 or 120)