

Some commonly held beliefs about media over QUIC

- “Always giving priority to the newest groups first is optimal for live video”
- “Greedy algorithm that always sends the highest-priority available data is optimal”

Counterexample 1

Assume the following setting:

- The link has a steady bitrate of 3 Mbps.
- The payload has a steady bitrate of 2 Mbps.
- The groups are 5 seconds long.
- The playback buffer size is 10 seconds long.

At some point, connection gets interrupted (no traffic comes through for some time), and when it's back, the subscriber has only two seconds worth of data left:



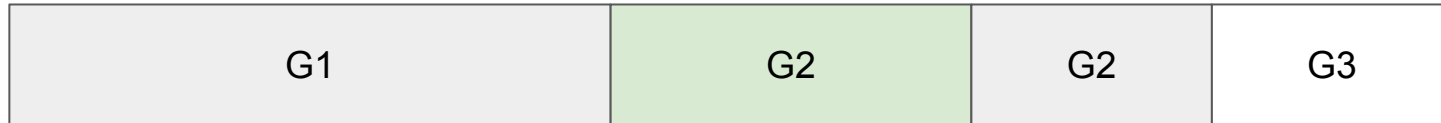
Counterexample 1: newest vs oldest



Initial state



2s after (oldest first)



2s after (newest first)

Counterexample 2

Assume the following setting:

- The publisher sends an audio-video stream to relay, it has 1 Mbps base video, 1 Mbps enhancement video, infinitely small bitrate audio.
- The subscriber to that relay has about 1.4 Mbps of bandwidth.
- Due to how the publisher's uplink works, the data arrives in fairly large (50ms) bursts.
- The problem: if the relay naively fills the link to the subscriber with all data it has in priority order, it will send enhancement streams, fill up congestion window, so when new audio arrives, it has to delay that.

MoQT Priorities

How should we approach them?

MoQT Use Cases

MoQT has one use case:

Delivering data that is generated
on-the-fly in a timely manner.

Question: What about VOD?

Short answer: **No.**

Long answer: MoQT's selling point is ability to deliver live-or-realtime media. VOD support is nice-to-have, but we shouldn't try to implement the "nice to have" features before we solve the actual hard problem.

Long short answer: VOD is generally easier than low latency use cases, so if we solve live/realtime, we should be able to do VOD too.

MoQT Use Cases

“Delivering data that is generated on-the-fly in a timely manner.”

Two important points that makes this hard:

- “Timely manner” is hard on unpredictable networks.
- “On-the-fly” means we’re not guaranteed to know when the data that we’re going to forward will be available to us.

There are also variable settings in which this needs to be achieved:

- Different end-to-end latency targets (10ms ~ 30s)
- Different networks
- Different payload profiles

Why MoQT Sending discussion is confusing

Currently, we have multiple proposals that look roughly this way:

- The subscriber or the publisher communicates some information to the relay.
- The relay takes some clearly defined actions based on this information.
- The combination of some or all of these mechanisms provides relay with clear instructions on how to forward things that would hopefully result in optimal delivery performance.

This approach is essentially **a kitchen sink**. We add mechanisms that relays have to follow to a T, and then hope that those make things better.

Alternative approach: stepping back

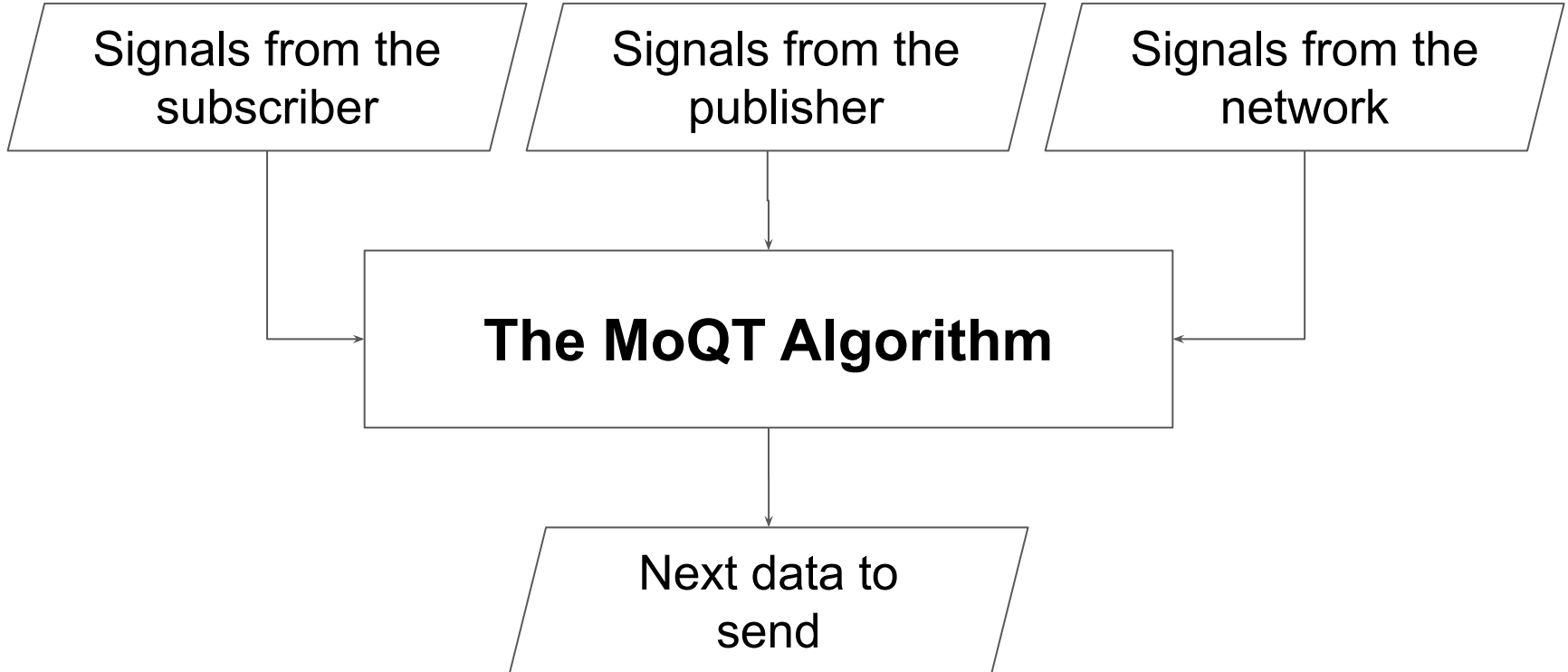
The fundamental question faced by MoQT senders is:

What is the data that I should send and when?

Note the similarities with the fundamental question of congestion control (which is “when should I send data next?”).

This is, fundamentally, a fairly challenging research problem.

MoQT sending algorithm: the diagram



MoQT sending algorithm: the outputs

The most general form of output: “decision of what to send when”.

Due to practical considerations, in some implementations the output space would be constrained by the capabilities of the QUIC API being used.

Possible “reduced” output spaces include priorities and timeouts.

MoQT sending algorithm: the inputs

What information does the relay need from the subscriber to make sending decisions?

What information does the relay need from the original publisher to make sending decisions?

How do we communicate that information?

MoQT sending algorithm: the algorithm

Given that this is “congestion control but harder”, we probably won’t arrive to one unique solution as a working group.

We should probably specify some baseline algorithm that does not do obviously wrong things.

Specific proposal

We should focus on what “inputs” we’re putting into the protocol.

The focus here is on “what”, because the “how” is complicated.

E.g. instead of specifying a delivery timeout, we provide relay with enough information about client’s playback buffer so that it can compute the timeout itself (and then adjust it on the fly).

In other words, move from “imperative” to “declarative” annotations for relays

Specific proposal: Inputs

Recall what different kinds of scenarios we have:

- Different end-to-end latency targets (50ms ~ 30s)
- Different networks
- Different payload profiles

Should we tell the relay those things if we know them?