ANRW2026
Applied Networking Research Workshop

Tokens, Not Packets: Rethinking the Multipath QUIC Scheduling Interface

Hendrik Cech[DE], Patrick Bokelmann[DE], Nitinder Mohan[NL]

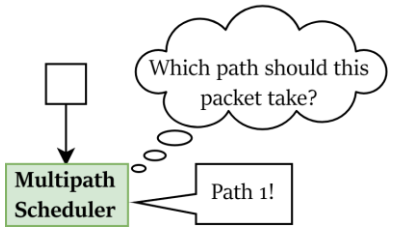
Technical University of Munich, Germany[DE]
Delft University of Technology, Netherlands[NL]

ACM/IRTF ANRW'26
2026-07-20
Vienna, Austria

TU Delft

Multipath Scheduling Model: The Problems

Multipath schedulers are usually defined and implemented as per-packet decision functions.

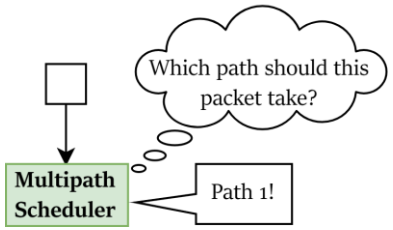


- Schedulers can only decide **where** to send packets, not **what** to send.
- Duplication, reinjection, etc., needs to hook into other parts of the protocol implementation.

→ P1: Tight coupling between scheduler and protocol implementation.

Multipath Scheduling Model: The Problems

Multipath schedulers are usually defined and implemented as per-packet decision functions.



- Schedulers can only decide **where** to send packets, not **what** to send.
- Duplication, reinjection, etc., needs to hook into other parts of the protocol implementation.

→ P1: Tight coupling between scheduler and protocol implementation.

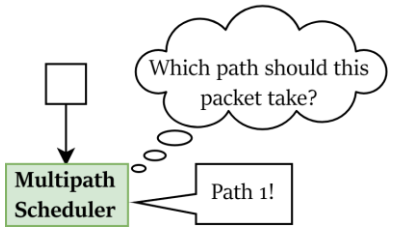
Multipath QUIC scheduling has many additional facets, compared to MPTCP:

1. **Reliable and unreliable delivery:** `STREAM` and `DATAGRAM` frames.
2. **Multiplexing**: multiple streams with different priorities (*stream-aware*).
3. **Control data scheduling:** control data carried in QUIC frames instead of TCP packet header fields.

→ P2: The per-packet scheduling model limits Multipath QUIC's capabilities.

Multipath Scheduling Model: The Problems

Multipath schedulers are usually defined and implemented as per-packet decision functions.



- Schedulers can only decide **where** to send packets, not **what** to send.
- Duplication, reinjection, etc., needs to hook into other parts of the protocol implementation.

→ P1: Tight coupling between scheduler and protocol implementation.

Multipath QUIC scheduling has many additional facets, compared to MPTCP:

1. **Reliable and unreliable delivery:** `STREAM` and `DATAGRAM` frames.
2. **Multiplexing**: multiple streams with different priorities (*stream-aware*).
3. **Control data scheduling:** control data carried in QUIC frames instead of TCP packet header fields.

→ P2: The per-packet scheduling model limits Multipath QUIC's capabilities.

→ P3: The app-transport boundary is difficult to cross: MPTCP is in-kernel; MPQUIC libraries offer no scheduler-specific API.

Design Choices

Design and implement a better Multipath QUIC scheduling interface:

1. **Uniform interface** that supports all mentioned facets of Multipath QUIC scheduling.
2. **Encapsulated scheduler:** no library-invasive integrations.
3. **Enable app-driven scheduling** to realize the full potential of Multipath QUIC.

Design Choices

Design and implement a better Multipath QUIC scheduling interface:

1. **Uniform interface** that supports all mentioned facets of Multipath QUIC scheduling.
2. **Encapsulated scheduler:** no library-invasive integrations.
3. **Enable app-driven scheduling** to realize the full potential of Multipath QUIC.

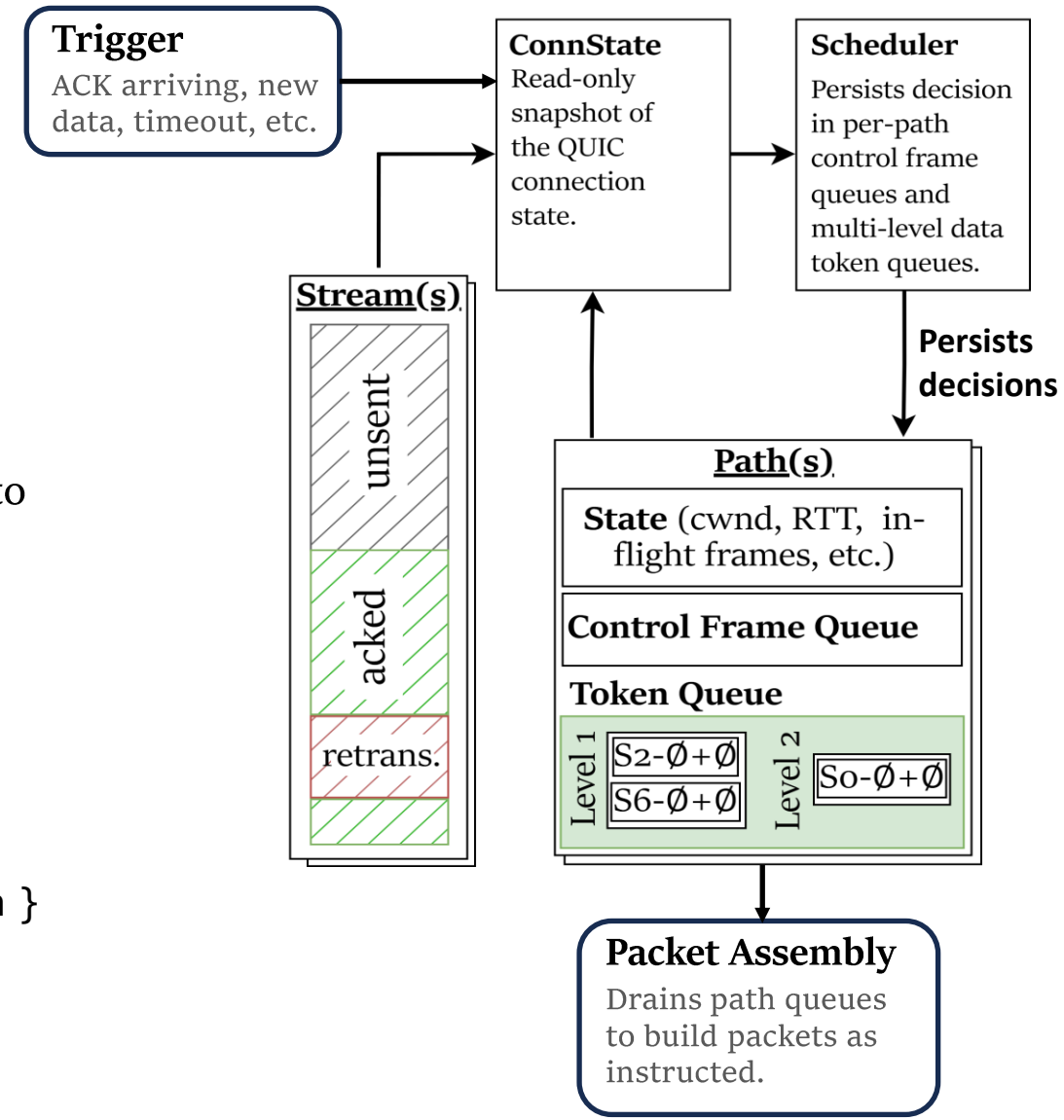
Design Choices:

1. **Up-front, inspectable scheduling**: the scheduler runs once before sending starts; it expresses all decisions as a persistent data structure.
2. **Mechanism-safe:** the library enforces protocol invariants; the scheduler never modifies QUIC state.
3. **Application-owned scheduler:** the application has direct scheduler access to enable easy app-guided scheduling.

Components & Invocation Model

We introduce five new components:

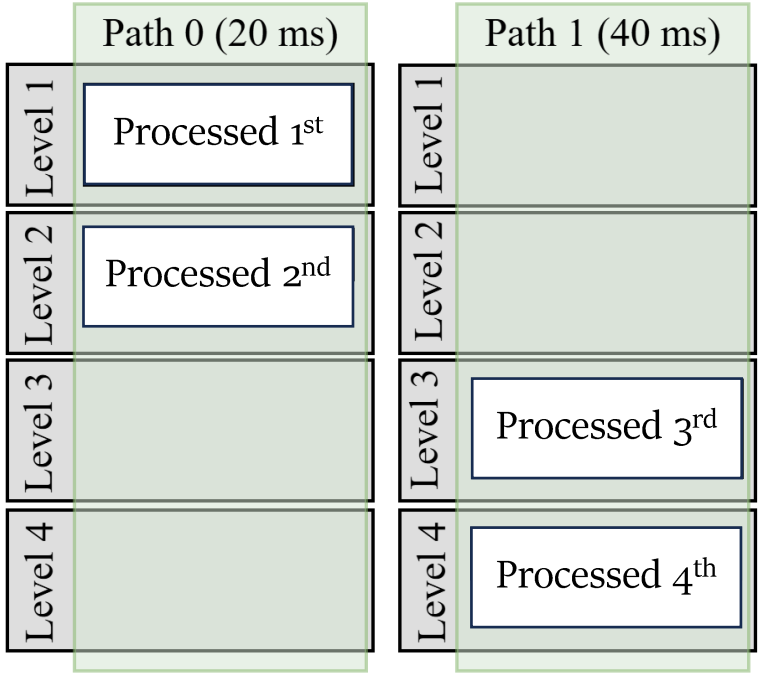
1. **ConnState**: Snapshot of the QUIC connection state, including pending control frames.
2. **Scheduler**: A function from ConnState to control frame and data assignments.
3. Per-path **control frame queues**.
4. Per-path **data token queues** with multiple *levels*.
5. **Data tokens:** express scheduling decisions translated.
   - `STREAM { stream_id, offset, length }`
   - `DATAGRAM {count }`



Worked Example: Stream-Aware MinRTT Scheduler

Example connection: two high-priority streams (#2 and #10) and a lower-priority stream (#8).

Multi-level Token Queues

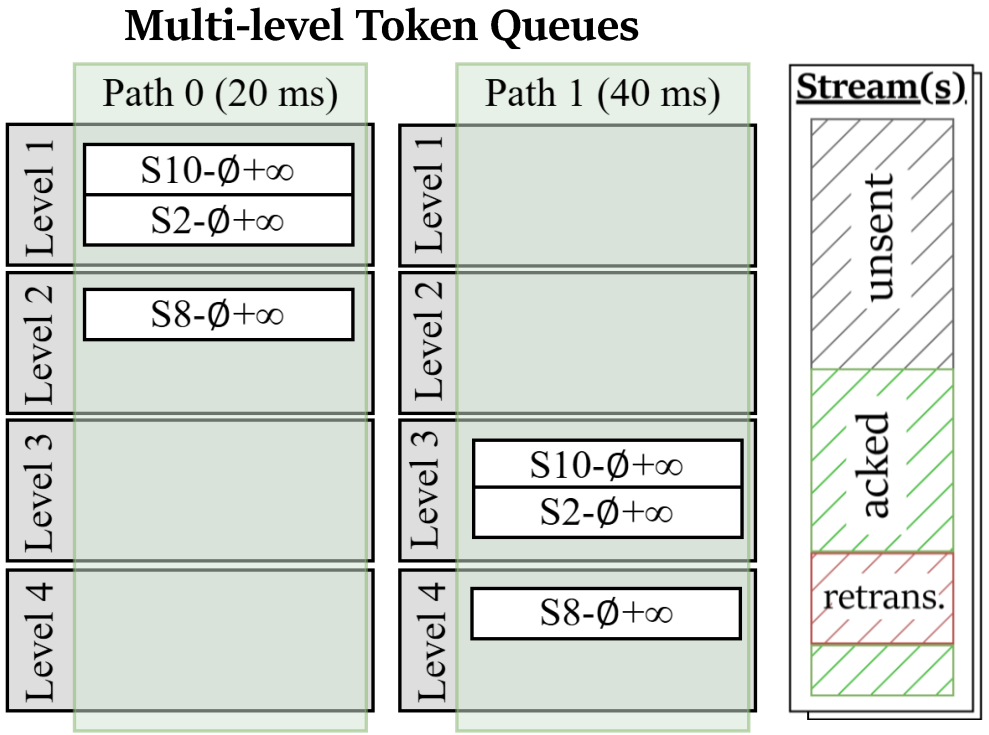


Processing order expressed through **levels**:

- MinRTT scheduling → path order
- Stream-aware scheduling → stream order

Worked Example: Stream-Aware MinRTT Scheduler

Example connection: two high-priority streams (#2 and #10) and a lower-priority stream (#8).



Processing order expressed through **levels**:

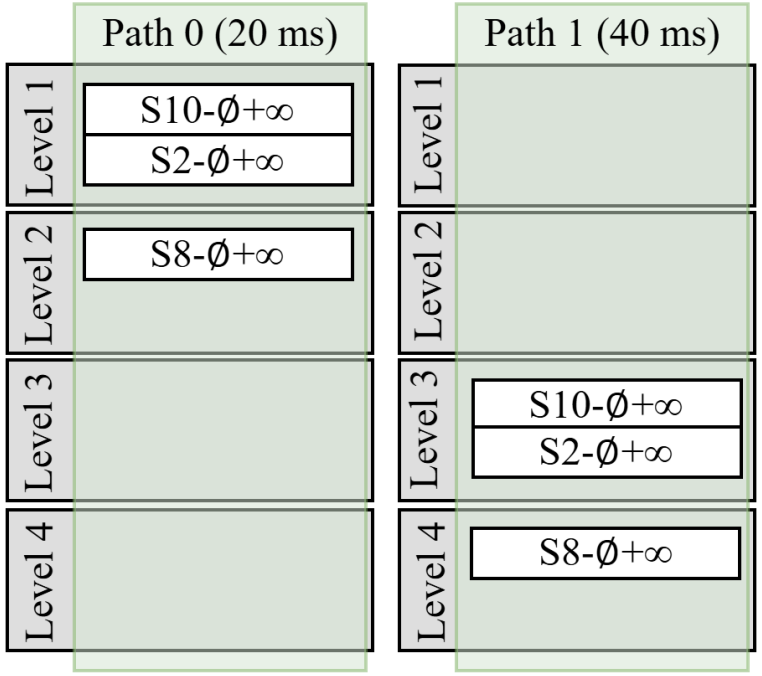
- MinRTT scheduling → path order
- Stream-aware scheduling → stream order

Levels contain **`STREAM` tokens** with three fields:

- specific **`stream_id`**
- **`offset`** = ∅: follow the library's send queue (retransmission or new data)
- **`length`** = ∞: persistent until stream is finished

The **queues are persistent**: only rebuilt if new streams are opened or if the path RTT order changes.

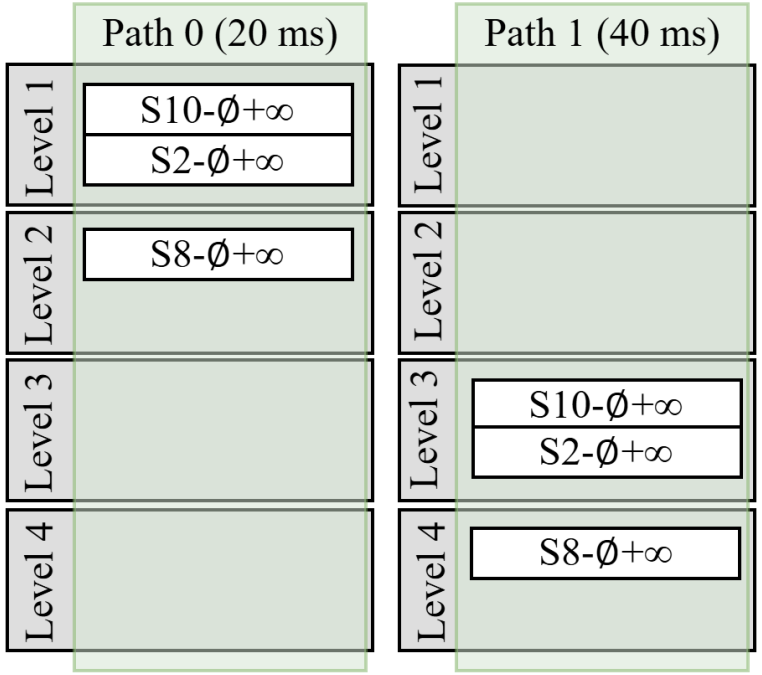
MinRTT Scheduler: Packet Generation



QUIC State
- S2: 1000B queued
- S8: 5000B queued
- S10: 8000B queued
- P0 cwnd_av: 3000B
- P1 cwnd_av: 1000B

MinRTT Scheduler: Packet Generation

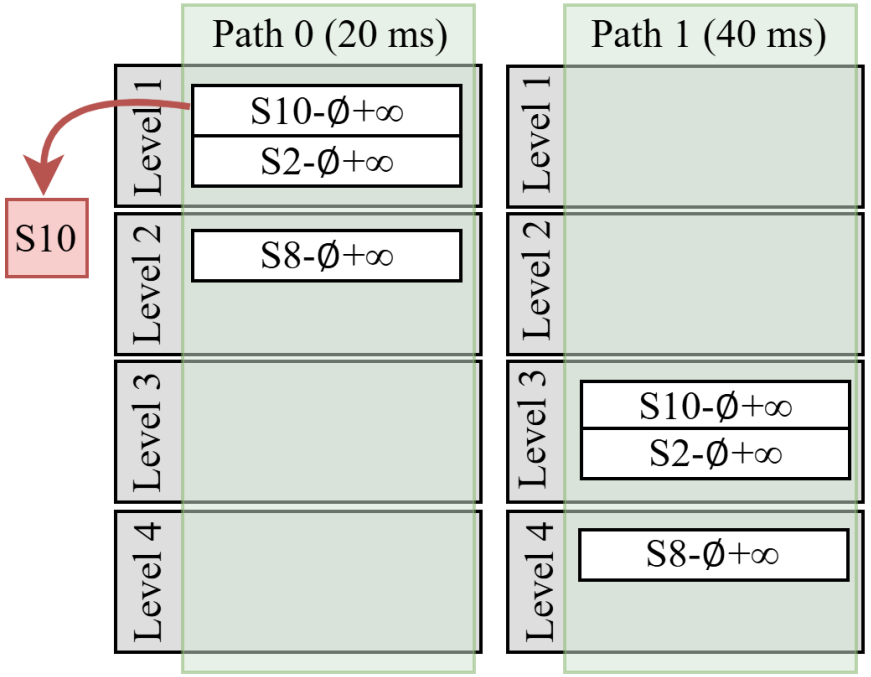
Processing rule:
The front token from the lowest available level is processed first.



QUIC State
- S2: 1000B queued
- S8: 5000B queued
- S10: 8000B queued
- P0 cwnd_av: 3000B
- P1 cwnd_av: 1000B

MinRTT Scheduler: Packet Generation

Processing rule:
The front token from the lowest available level is processed first.

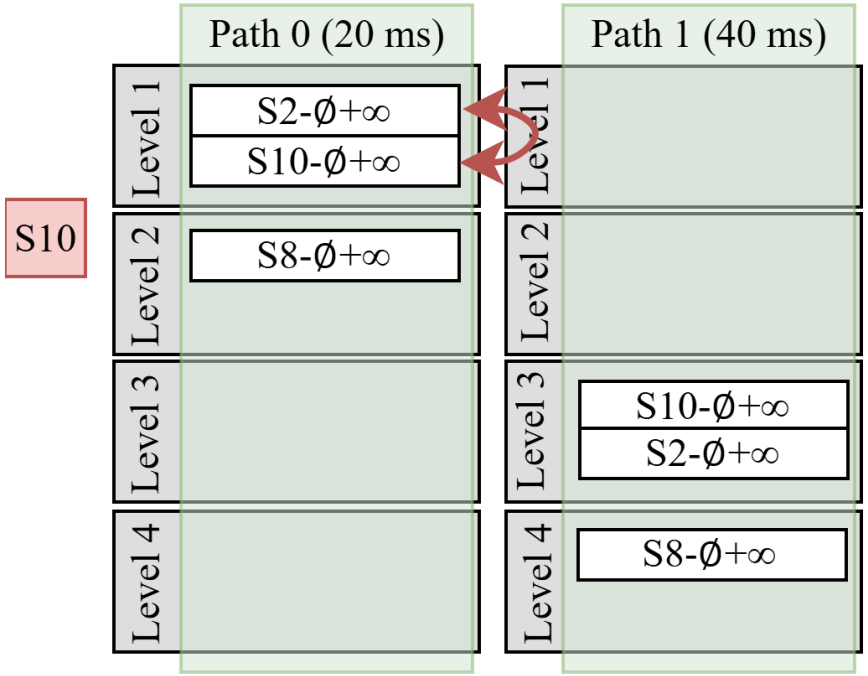


QUIC State
- S2: 1000B queued
- S8: 5000B queued
- S10: 7000B queued
- P0 cwnd_av: 2000B
- P1 cwnd_av: 1000B

Front token in lowest level processed
→ QUIC STREAM frame on path 0 emitted (stream 10), filling the packet

MinRTT Scheduler: Packet Generation

Processing rule:
The front token from the lowest available level is processed first.

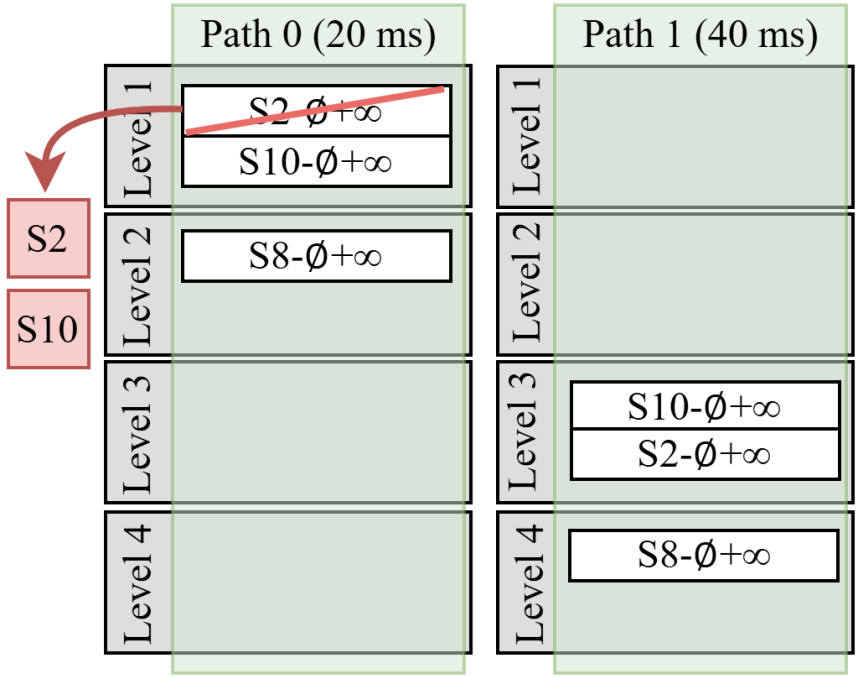


QUIC State
- S2: 1000B queued
- S8: 5000B queued
- S10: 7000B queued
- P0 cwnd_av: 2000B
- P1 cwnd_av: 1000B

Processed token is requeued at the end of its level (`length` = ∞)

MinRTT Scheduler: Packet Generation

Processing rule:
The front token from the lowest available level is processed first.

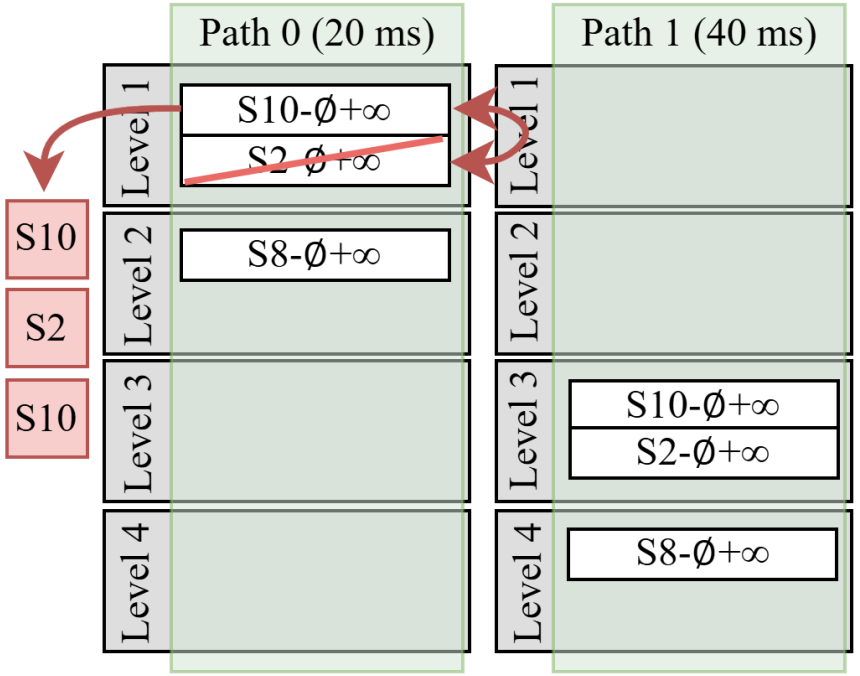


QUIC State
- S2: 0B queued
- S8: 5000B queued
- S10: 7000B queued
- P0 cwnd_av: 1000B
- P1 cwnd_av: 1000B

Front token in lowest level processed
→ QUIC `STREAM` frame on path 0 emitted (stream 2)
→ Token disabled since no more bytes are queued

MinRTT Scheduler: Packet Generation

Processing rule:
The front token from the lowest available level is processed first.

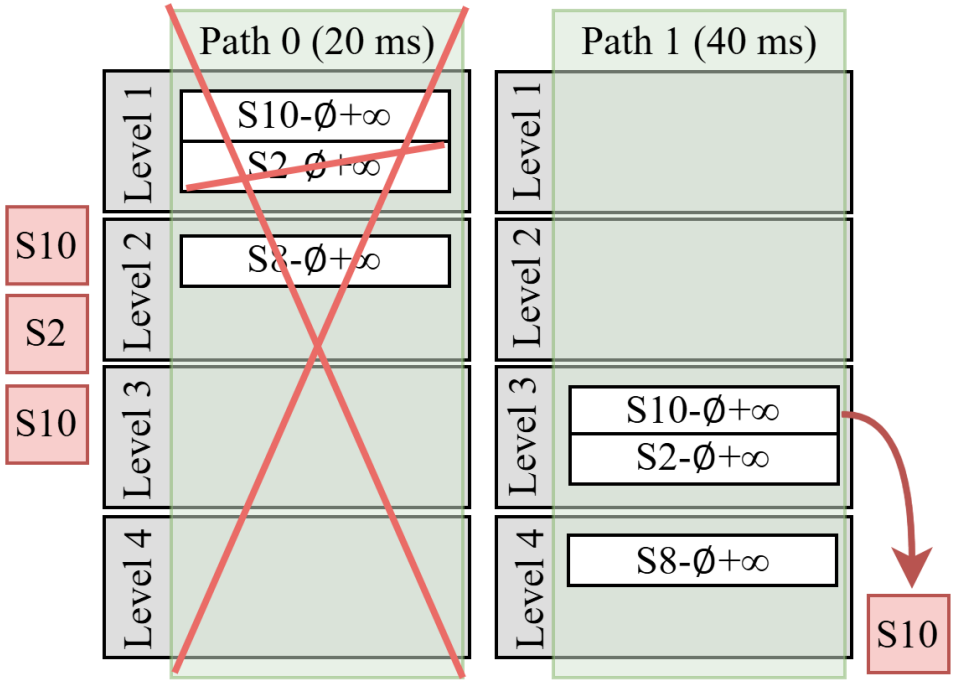


QUIC State
- S2: 0B queued
- S8: 5000B queued
- S10: 6000B queued
- P0 cwnd_av: 0B
- P1 cwnd_av: 1000B

Processed token (S2) is requeued at the end of its level (`length` = ∞).
→ QUIC `STREAM` frame on path 0 emitted (stream 10)
→ Token disabled since no more bytes are queued

MinRTT Scheduler: Packet Generation

Processing rule:
The front token from the lowest available level is processed first.

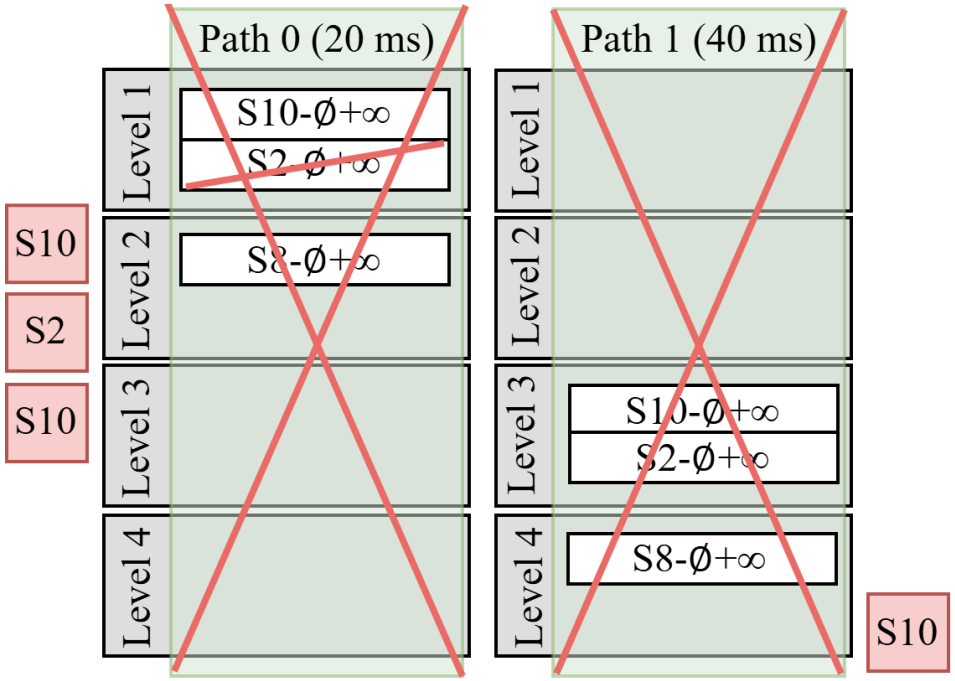


QUIC State
- S2: 0B queued
- S8: 5000B queued
- S10: 5000B queued
- P0 cwnd_av: 0B
- P1 cwnd_av: 0B

Path 0 is disabled since its congestion window is exhausted.
First available token is processed: S10 token on level 3 on path 1.
→ QUIC `STREAM` frame on path 1 emitted (stream 10)

MinRTT Scheduler: Packet Generation

Processing rule:
The front token from the lowest available level is processed first.



QUIC State
- S2: 0B queued
- S8: 5000B queued
- S10: 5000B queued
- P0 cwnd_av: 0B
- P1 cwnd_av: 0B

Path 1 is disabled since its congestion window is exhausted.
→ Packet generation stops since all tokens and paths are disabled.
→ QUIC sleeps until the next trigger arrives.

MinRTT is just about the simplest scheduler there is, but THIS wasn't simple …

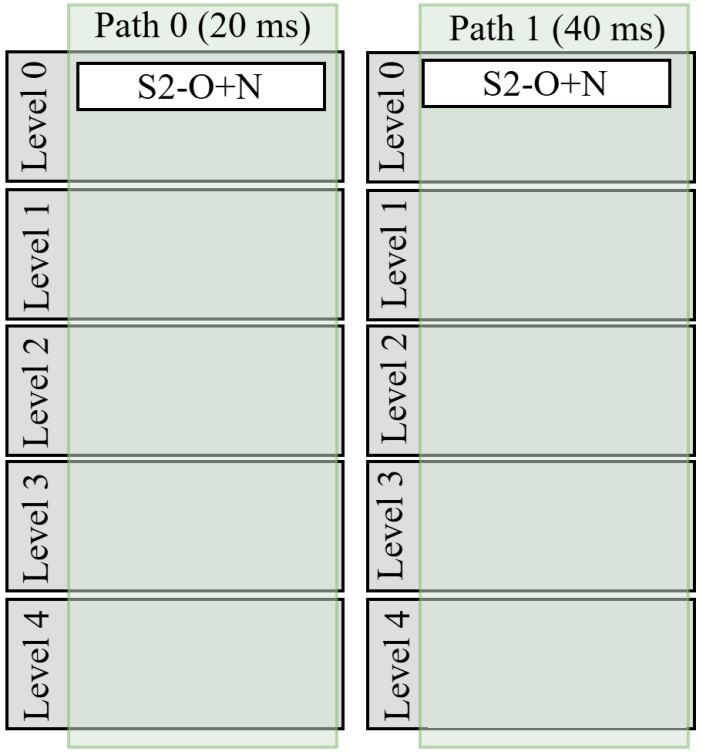
What's the point?

Small Changes, Mighty Consequences

Tweaking the MinRTT scheduler to support …

1. **Redundant retransmissions**: lost range [O, O+N)? Push tokens with `offset=O`, `length=N` bytes at level 0 on all paths.

Multi-level Token Queues



Small Changes, Mighty Consequences

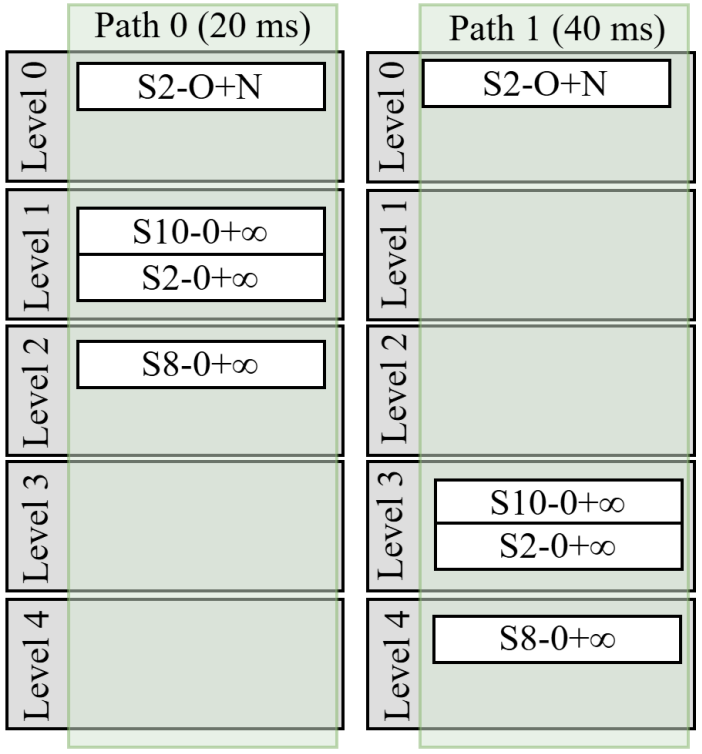
Tweaking the MinRTT scheduler to support …

1. **Redundant retransmissions**: lost range [O, O+N)? Push tokens with `offset=O`, `length=N` bytes at level 0 on all paths.
2. **Full redundancy**: change offset `∅` → `0`: each token starts at byte 0, emits N bytes, requeues with `offset+=N`. ACKed bytes are skipped.

New scheduler capability: **control frame routing**.

- Every pending control frame must be assigned to a path
- Control frames can be **scheduled:** towards the fastest path, duplicated to multiple paths, etc.

Multi-level Token Queues



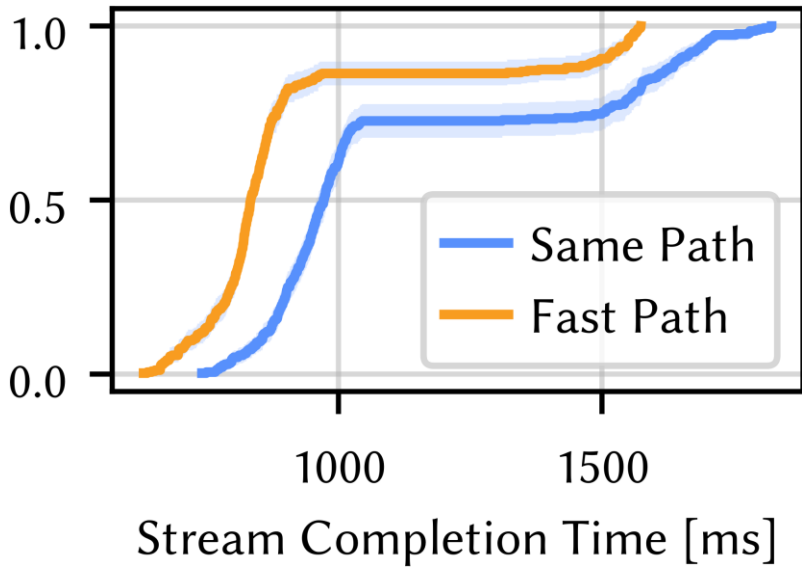
Evaluation

- **Implementation:** Open-source prototype based on Cloudflare's quiche.
- **Mininet environment**: 2 × 10 Mbps, 10 / 25 ms paths
- **Workload**: 100 KB over 16 streams (repeated 30 times)
- **Metric**: Stream Completion Time (SCT)

Fixed Scheduler (MinRTT), different **ACK routing policies**:

- Same Path: return ACKs on the receiving path
- Fast Path: all ACKs on the lowest-RTT path

→ **Reduced median SCT by 138 ms**



Conclusion: Tokens, Not Packets

Strengths of our model:

- Clean encapsulation of scheduling logic
- Potential for portable scheduler implementations
- Inspectability of scheduling policy eases development and testing

Limitations:

- No control over packetization
- Choices of triggers limit granularity of scheduling decisions

Where to go from here?

- Support other MPQUIC libraries
- Develop a validation test suite to prove portability of scheduler between MPQUIC substrates
- Conduct fair head-to-head scheduler and implementation study

Tokens, Not Packets: Rethinking the Multipath QUIC Scheduling Interface

Hendrik Cech[DE], Patrick Bokelmann[DE], Nitinder Mohan[NL]

Technical University of Munich, Germany[DE]
Delft University of Technology, Netherlands[NL]

ACM/IRTF ANRW'26
2026-07-20
Vienna, Austria

Paper

https://hendrikcech.com/documents/cech2026tokens.pdf

Code

https://github.com/hendrikcech/anrw26-tokens-not-packets

Architecture

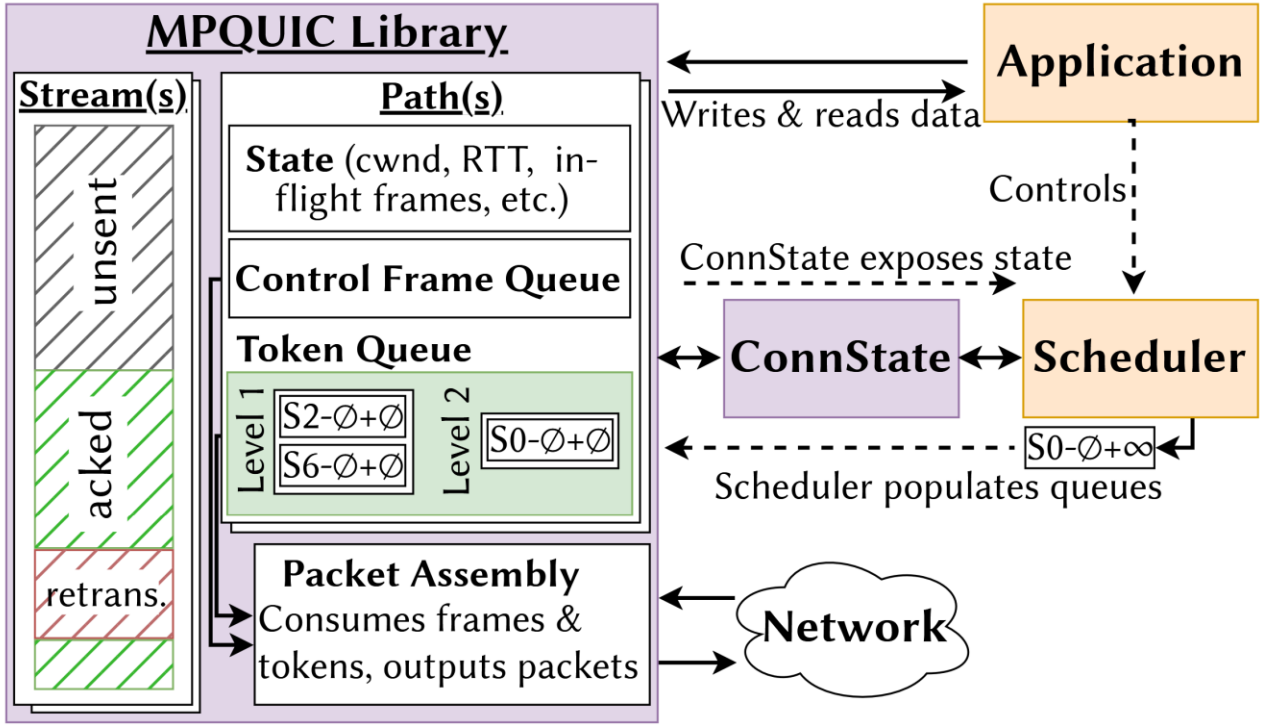


Figure 1: The *ConnState* object mediates access of the application-controlled scheduler to the QUIC state.

Scheduler Taxonomy

Table 1: MPQUIC/MPTCP schedulers span two largely orthogonal axes (invocation mode and decision granularity) with a third emerging axis (information source). A uniform interface must accommodate all three.

Scheduler	Mode	Granularity	Info. source
ECF [17]	per-packet	path	transport
BLEST [6]	per-packet	path	transport
STTF [13]	per-packet	path	transport
Peekaboo [30]	per-packet	path	transport
QAware [28]	per-packet	path	below-transport
STORM [11]	per-packet	path	below-transport
SA-ECF [25]	per-packet	stream	app (priority)
Monty [24]	per-packet	stream	app (utility)
XLINK [31]	per-packet	stream range	app (QoE)
Chorus [19]	per-packet	stream range	app (ABR)
DAPS [15]	up-front	path	transport
MuSher [26]	up-front	path	transport
PStream [27]	up-front	stream	app (priority)
SR-MPQUIC [20]	up-front	stream range	app (priority)
EdAR [9]	up-front	stream range	transport

Token Definitions

Table 2: The six modes of a `Stream` token with a fixed `stream_id`. Symbol ∅ denotes the empty value.

`offset`	`length`	Behavior
∅	None	Emit one STREAM frame from the lowest queued offset; drop.
∅	N	Emit up to N bytes from the lowest queued offset; requeue with decremented `length`, drop at 0.
∅	∞	Emit from the lowest queued offset; always requeue until stream termination (used by MinRTT).
O	None	Emit one STREAM frame starting at O; drop.
O	N	Emit frames covering $[O, O+N)$; if bytes at O have already been ACKed, `offset` advances to the smallest buffered offset below $O+N$. Requeue with updated `offset`/`length`, drop on $N = 0$.
O	∞	Traverse the stream from O; `offset` advances past ACKed bytes; persists until stream terminates.

Table 3: The behavior of the `Datagram` token depending on its `count` parameter.

`count`	Behavior
N	Emit up to N datagrams from the send queue; requeue with decremented `count`, drop at 0.
∞	Emit a datagram if the send queue is non-empty; always requeue until connection termination.

Runtime Performance

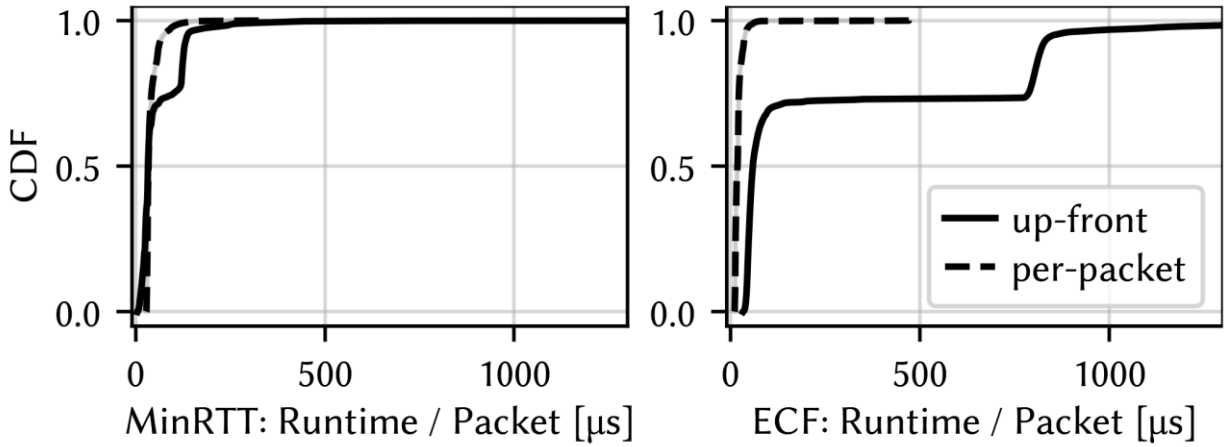
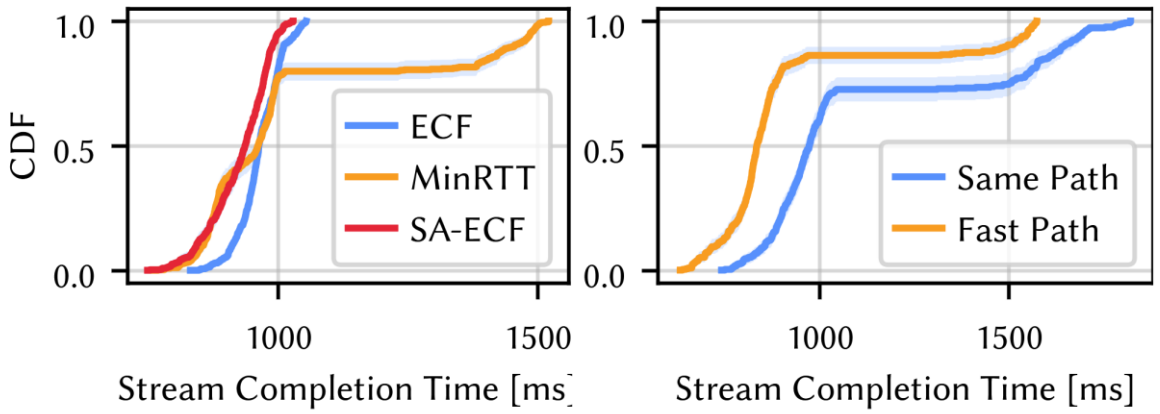


Figure 3: The runtime of per-packet and up-front implementations normalized by emitted packets.

Evaluation



(a) (SA)ECF. (b) MinRTT ACK policies.

Figure 4: Demonstration of up-front (a) STREAM and (b) control frame schedulers.