Datagram v.s. Stream
Experience sharing from Implementing DoQ

Chuan Ma

Yixin Liao, Hang Shi

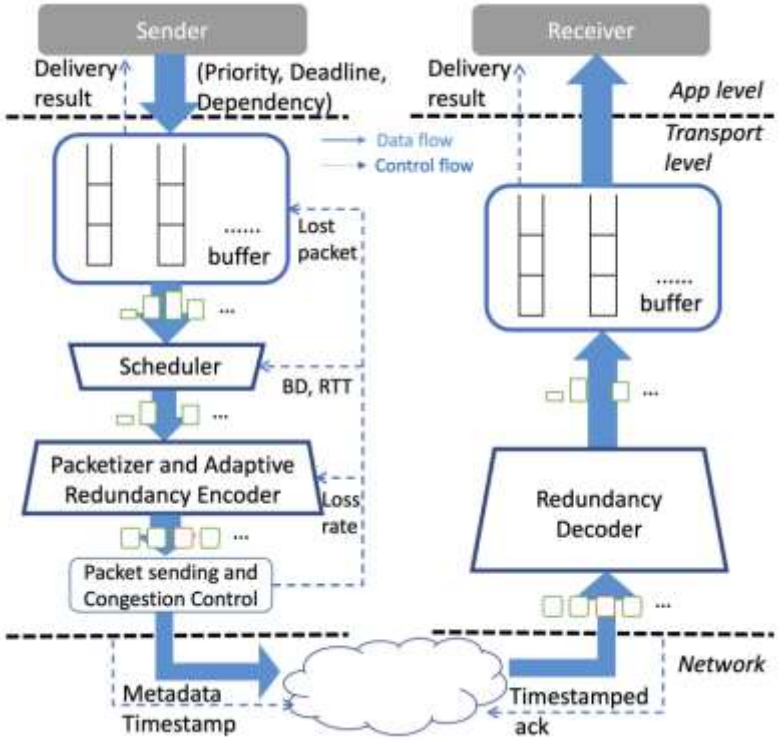
January 31, 2023

Table of Contents

- Introduction of DTP
- From DTP to MoQ: DoQ
- Implementation
- Test and Results
- Discussion
- Summary
- Future Works

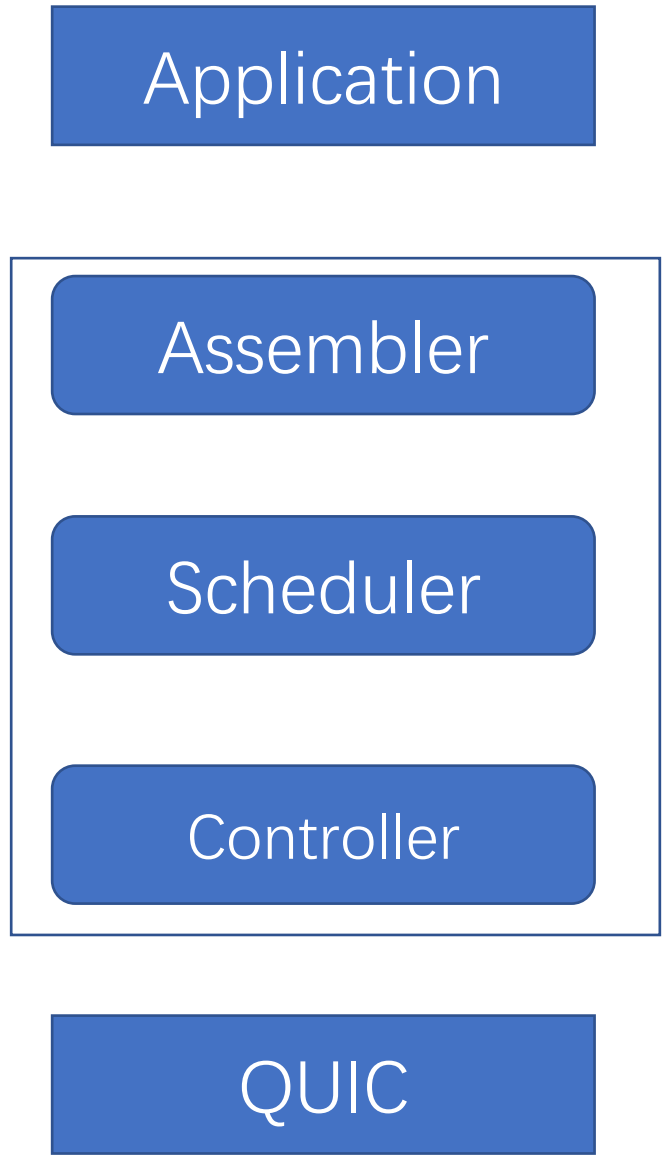
Introduction of DTP

- DTP: Deadline-aware Transport Protocol
- Keywords
  - Block: A independent chunk of data, like video frame
  - Deadline: Overdue data may be dropped by the App
- An Extension of QUIC for media transport
  - Block based sending: Matching Block to QUIC stream
  - New Frames: Sending Block metadata
  - Deadline-aware Scheduler and Block Canceller
  - Adaptive Redundancy Encoder
- Use cases: VR/AR, Live Stream, online gaming …
- Offer deliver before deadline service to application
- Draft: https://datatracker.ietf.org/doc/draft-shi-quic-dtp/



From DTP to MoQ: DoQ

- MoQ Charter: *This working group will not propose changes to the underlying QUIC transport*
- Realize Deadline-aware functions over QUIC
- DoQ: Deadline-aware over QUIC
- Designs
  - Layered Structure
  - Stream/Datagram transmission
  - Assembler: Block Segmentation/Reassemble
  - Scheduler: Block/Segment level scheduler
  - Controller: handle control flow and feedback



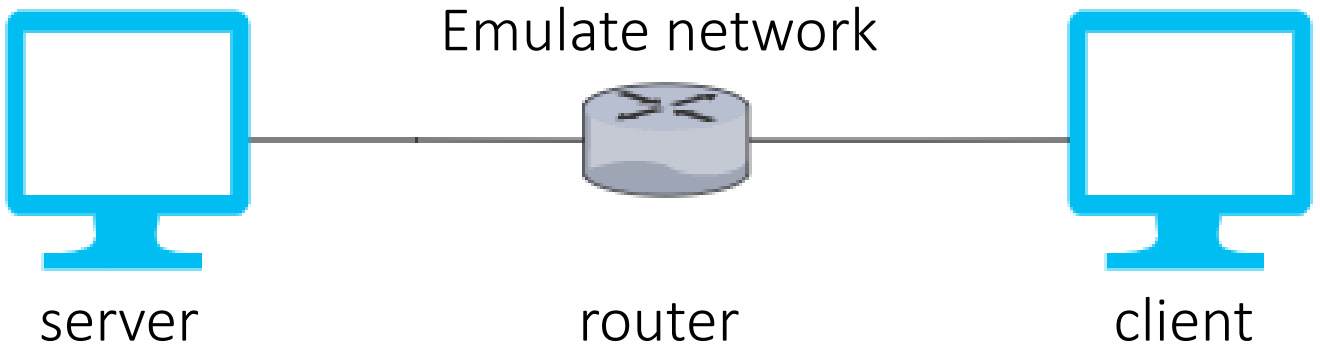
Implementation

- Built on top of quiche[1]
  - Enable stream priority, QUIC datagram and multipath
- Supports Stream mode and Datagram mode
  - Stream: Match each block to single unidirectional stream
  - Datagram: Break blocks into segments to send and receives
- Metadata
  - General: block_id, block_size, priority, deadline, create_time, send_time
  - Datagram only: offset, data_len
  - Stream at the head, Datagram at each datagram's head
- Currently not support relay and datagram retransmission

1. Cloudflare/quiche: https://github.com/cloudflare/quiche

Test Setting

- A simple End-to-End transmit test
- Data trace from an online meeting program
- Use traffic control to emulate different network conditions
- No relay, No datagram retransmission, No stream cancelling
- Stream mode v.s. Datagram mode

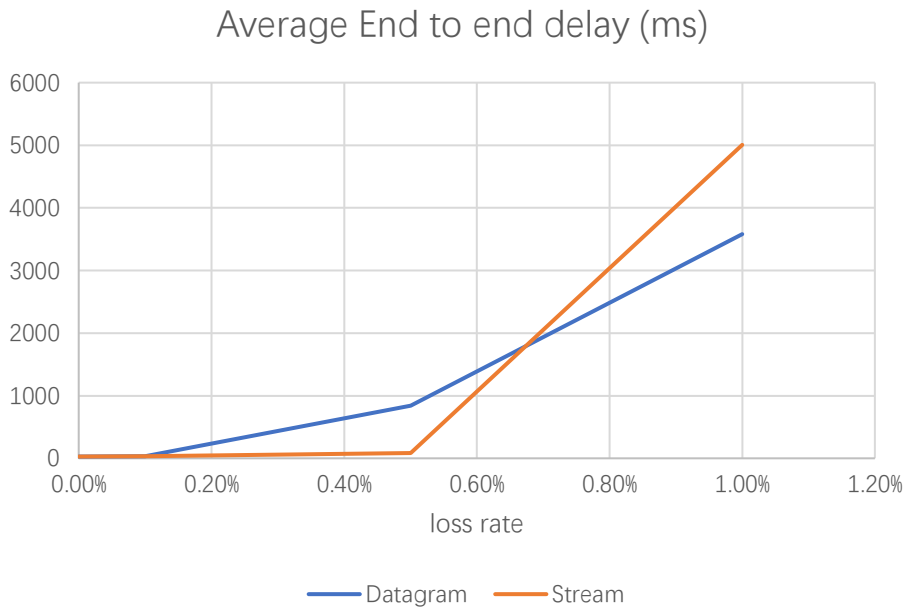


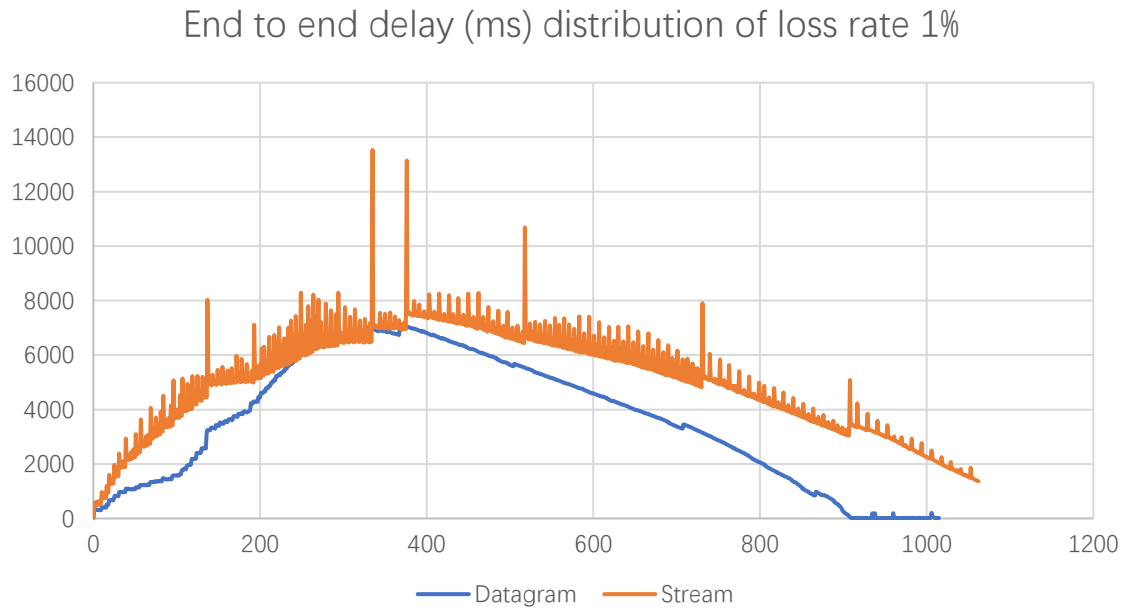
Test Setting

- End to end delay
  - Time when the block is finished – Time when the block is created
- Finish condition
  - A block is either finished or dropped
- Stream mode block finish condition
  - The block is completed received by the application
- Datagram mode block finish condition
  - The block is completed received by the application
  - The block is overdue and has been received more than 90% of data

Test Results: Influence of packet loss

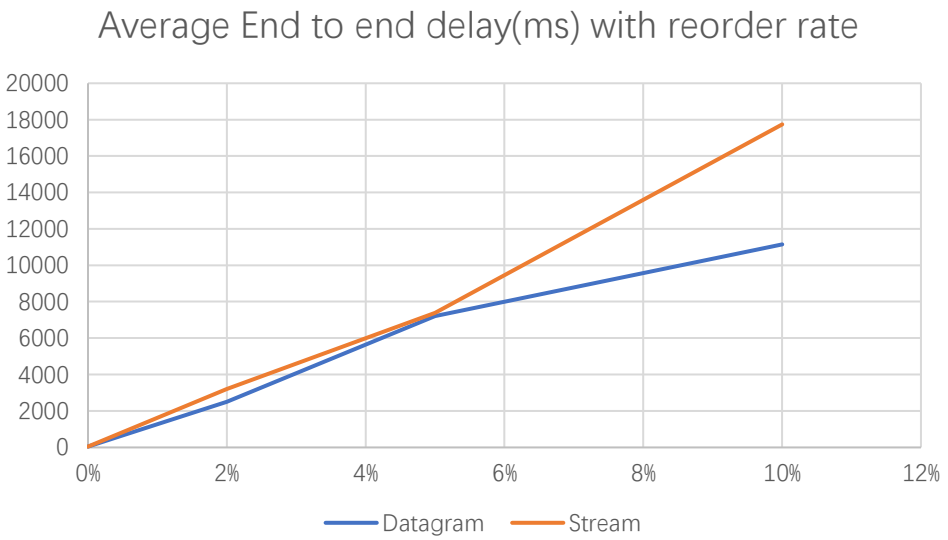
- Network: RTT 50ms, reorder 0%
- Datagram's Average E2E delay is significantly larger when loss rate < 1%
- Stream mode's Average E2E delay increases when loss rate > 1%

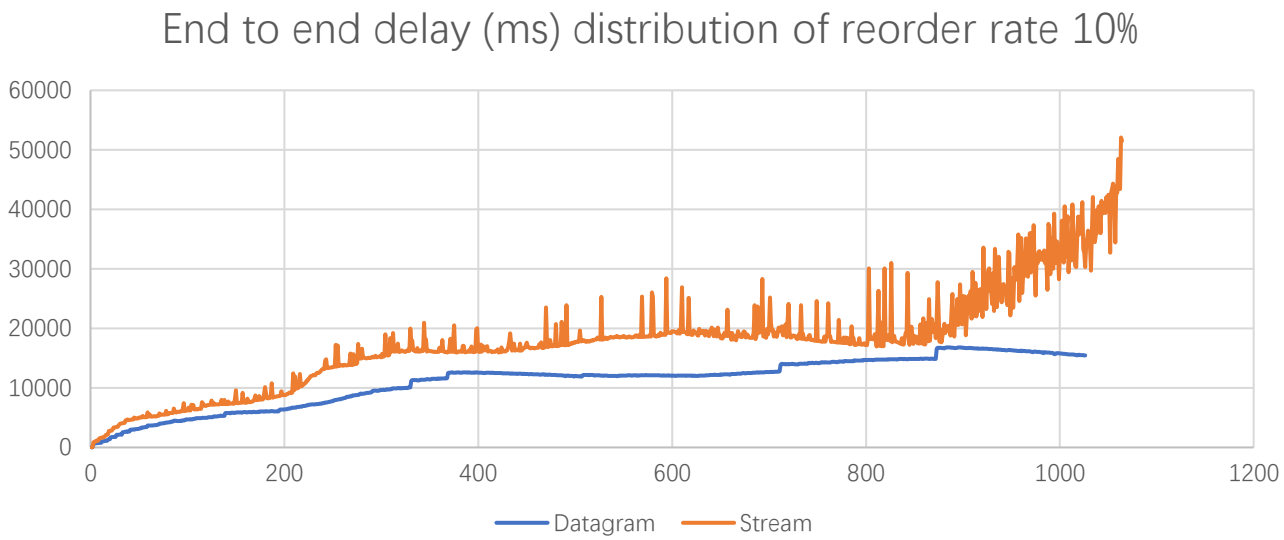




Test Results: Influence of packet reorder

- Network: RTT 50ms, loss rate 0%
- Both modes' Average E2E delays are similar when reorder rate < 5%
- Stream mode's Average E2E delay is larger when reorder > 5%

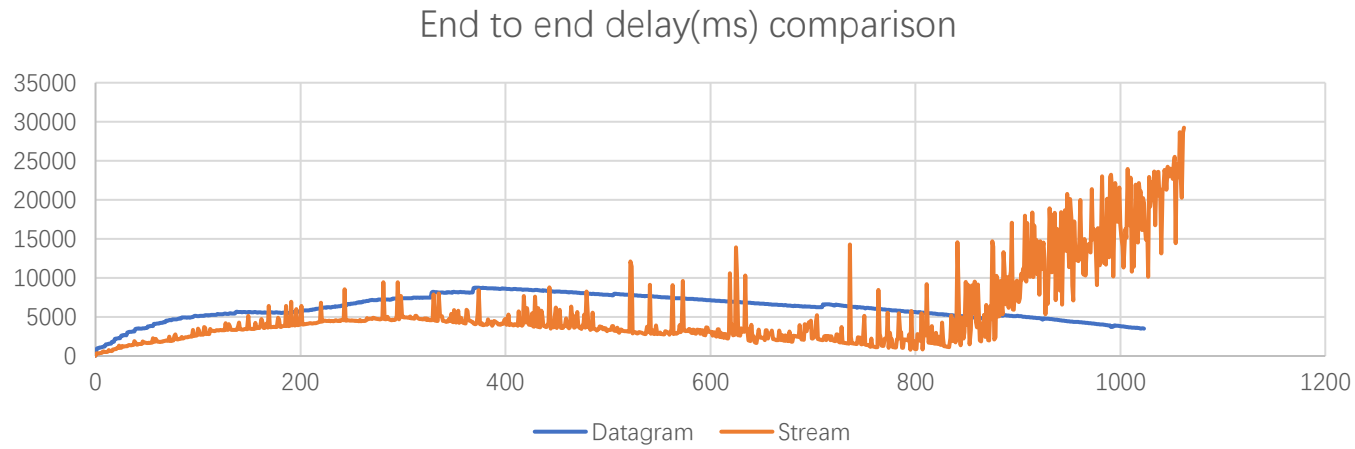




Test Results: Some observations

- Datagram mode may get incomplete block without retransmission
  - About 1% of block will loss more than 10% of data in loss rate 0.1%
  - Allow incomplete block can significantly decrease the total E2E delay
- Stream mode's E2E delay will jitter obviously when packet loss or reorder
  - Retransmitting old blocks obstructs new blocks
  - Good retransmission and stream cancelling strategy helps

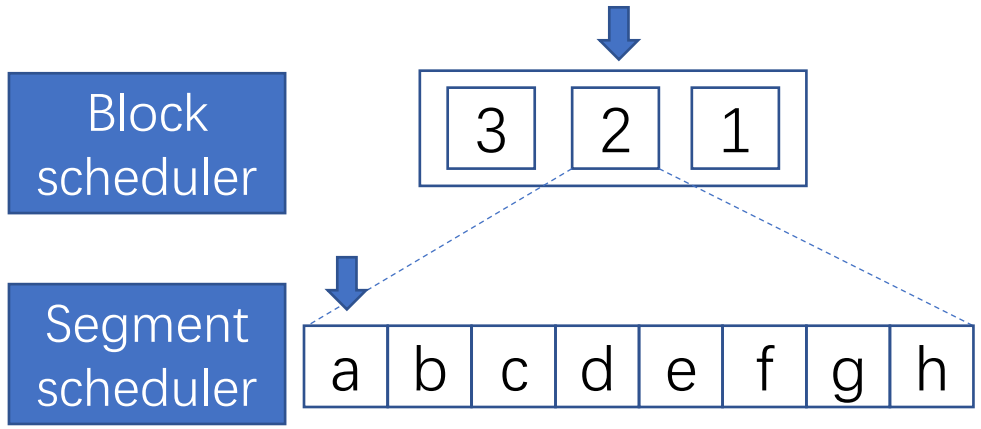
Network:
RTT 50ms,
loss rate 0.1%
reorder rate 5%



Discussion: Metadata

- Stream mode
  - Add header before the block data
- Datagram mode
  - Add header to each QUIC datagram
- Other ways to carry metadata ?
  - Through control stream in datagram mode?

Discussion: Scheduler

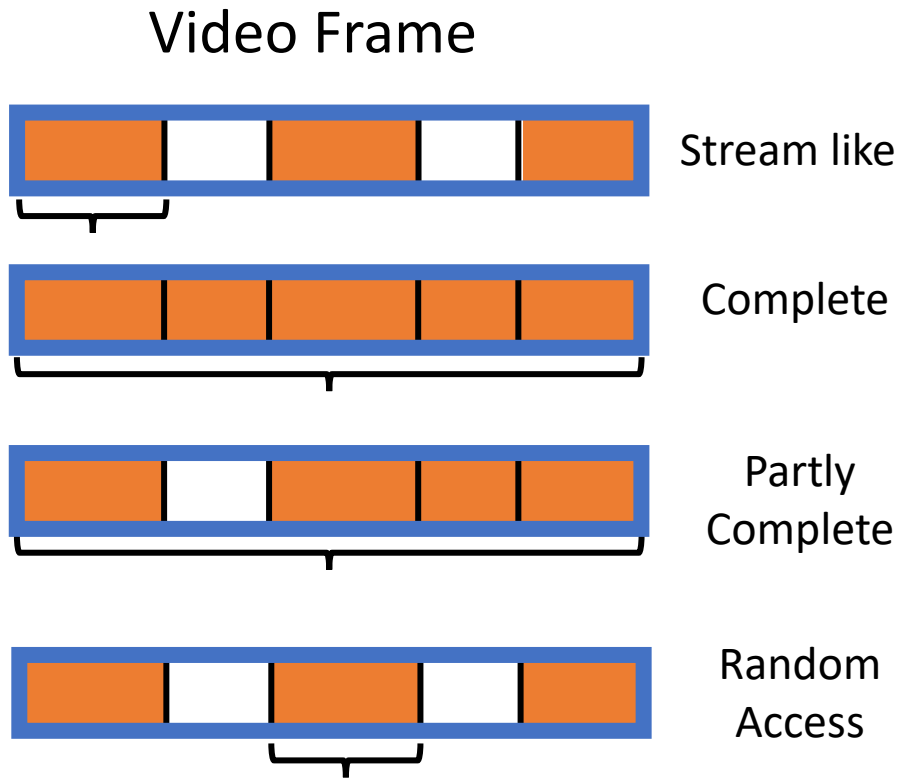


- Two main schedulers
  - Block scheduler
  - Segment scheduler
- Difference of granularity: block, stream or segment
  - quiche only offer stream level scheduling
- Stream implementations should be aware of the behavior of QUIC stream scheduler
- Datagram implementations should define their own schedulers

	Stream mode	Datagram mode
Block scheduler	Customizable	Customizable
Segment scheduler	Need QUIC support	Customizable

Discussion: API for Receiver Applications

- How should application acquire data from MoQ implementation?
- When should DoQ offer the data?
- Stream like: Head of line blocking
- Finish: Only when the frame is complete
- Partly Finish: e.x. 90% of the frame data
- Random Access



Discussion: Others

- Block (Stream) canceling
  - Overdue blocks significantly increase end-to-end delay of other blocks
  - Some algorithms to cancel block is essential
- Congestion Control
  - CC should be configurable for MoQ implementations
- Flow Control
  - Some QUIC implementations have flow control and may block the data
  - Flow Control may cause datagram to be dropped
- Retransmission
  - ACK or NACK or No Retransmission
  - Retransmit old block segment or new block segment

Summary

- We implement a basic MoQ system DoQ based on the designs of DTP
- Stream mode
  - Pros: Easy to implement, stable performance, …
  - Cons: Heavily restricted by the implementation of QUIC, …
- Datagram mode
  - Pros: Allow fine-grained customize control, have some advantages when packet loss or reorder, …
  - Cons: Possible complex in implementation, …
- Some key designs like the scheduler, the canceling mechanism need further discussion

Future Works

- Implement (before IETF 116)
  - Relay
  - Retransmission of datagram
- Drafts
  - Design of DoQ
  - Relaying of Deadline
- Discuss MoQ requirement of Deadline
- Any suggestion?

Thanks

Q&A, Discussion