Transport parameters for QUIC 0-RTT connections

draft-kuhn-quic-0rtt-bdp-07

Nicolas Kuhn, Emile Stephan, Gorry Fairhurst, Tom Jones

nicolas.kuhn@cnes.fr emile.stephan@orange.com
gorry@erg.abdn.ac.uk tom@erg.abdn.ac.uk

ICCRG IETF-110

This talk concerns a transport (QUIC) and a path characteristic (satellite)

- Paths can be very different characteristics
  - Higher delay >>10 mS to ~secs of Path RTT (~650 ms for GEO)
  - High capacity: Large Bandwidth Delay Product
  - Capacity available on demand (not *always* available)
  - Asymmetry improves overall efficiency
- Impacts
  - Delay: Startup; Flow Control Procedure
  - BDP: Flow Control Buffers; cwnd
  - Capacity: Not safe to assume always high capacity; but mostly true
  - Asymmetry: Watch-out for ACKs, etc

- Other paths might also have similar needs.

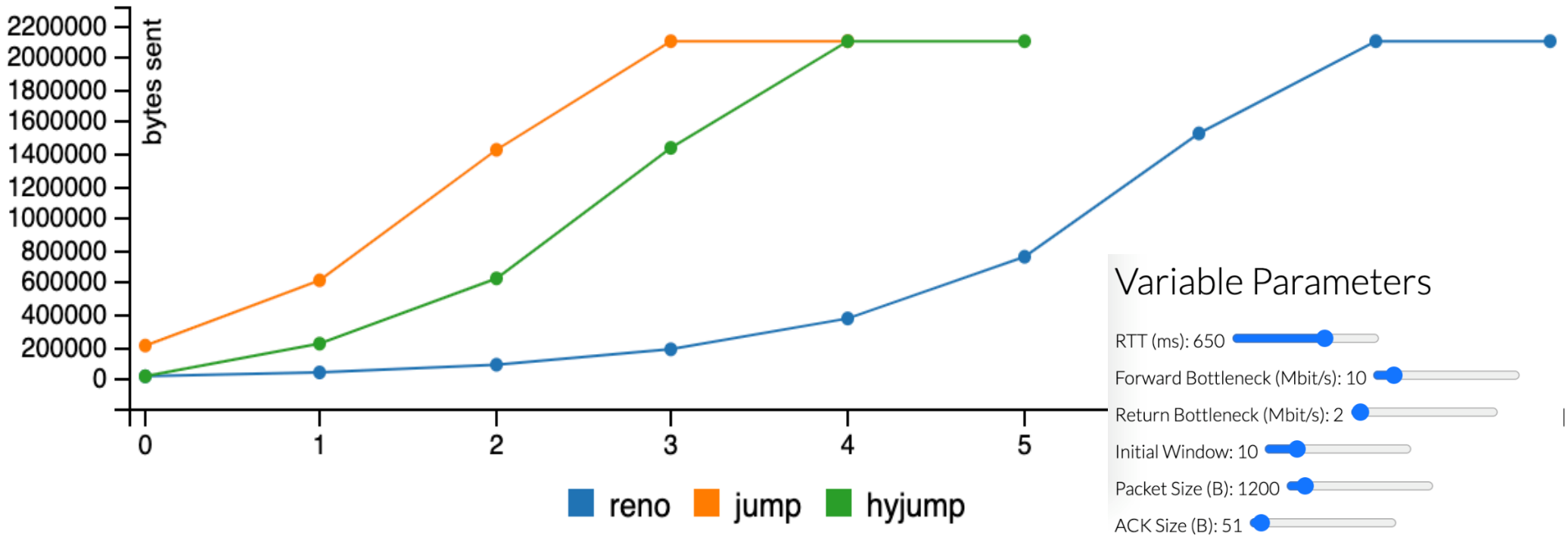
Context

- Extension to transport parameters
  - Shared during the 0-RTT phase (RTT, BDP, etc)
  - Allows resumption using the additional transport and connection properties discovered from previous connections
- Use cases:
  - Optimizing client requests
  - Safe jump in cwnd/flow control size
  - Sharing transport information across multiple connections

Similar idea proposed for H2 and TLS1.3 in :

"Optimizations for Using TLS Early Data in HTTP/2 ; draft-thomson-httpbis-h2-0rtt-00”

A very simple example of why this helps:



BDP & RTT have a large impact on growth
Reno: Effect of RTT
Jump: a simple jump to 25% of previous cwnd
Hyjump: More considered jump to 25% of cwnd and then Reno

New transport parameters for QUIC to help resume connections in 0-RTT mode

- First connection without 0-RTT:
  - Server stores parameters in BDP extension
  - At the end of the first connection:
    - Server sends the BDP extension frame to the client
    - Both client and server can read the content of the BDP extension

- Second connection with 0-RTT:
  - Both client and server can retrieve values stored in the BDP extension:
    - The client can recall them when reconnecting
  - Path to endpoint *could* have changed; capacity *could* have changed

BDP metadata

- recon_bytes_in_flight (0x000X):
  - The bytes in flight measured on the previous connection by server
- recon_min_rtt (0x000X):
  - The minimum RTT measured on the previous connection by server
- recon_max_pkt_number (0x000X)

With this information we can jump: How do we do so safely?

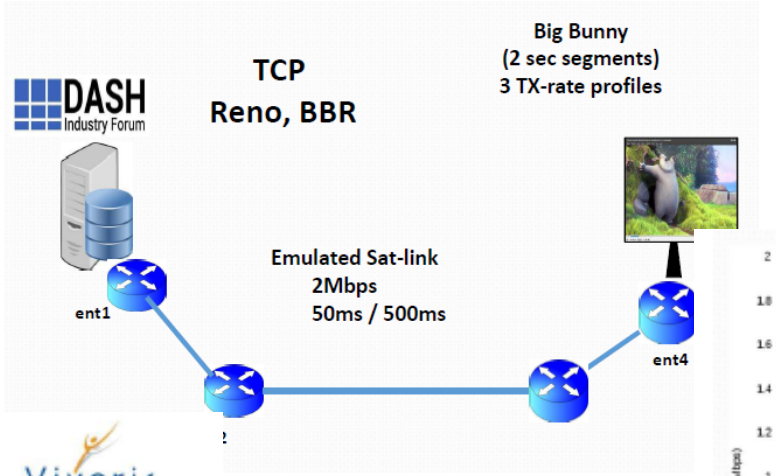
Motivation – Transport information across multiple connections

- Sharing transport information across multiple connections

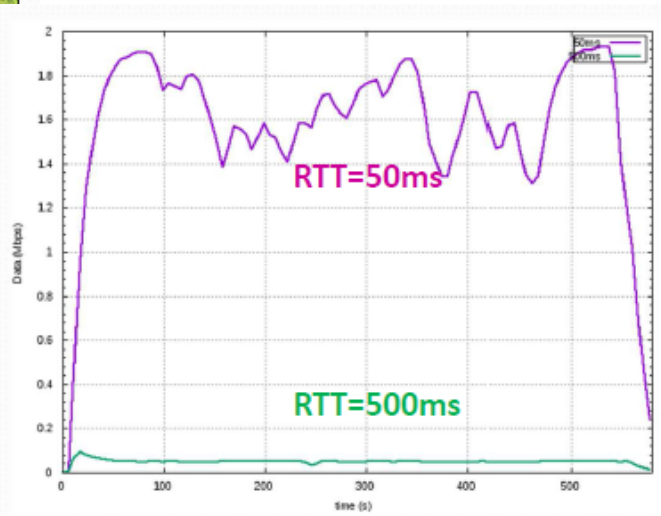
- See I-D.ietf-tcpm-2140bis
  - TCP Control Block Interdependence
  - draft-ietf-tcpm-2140bis-09.txt

Motivation - Optimizing client requests

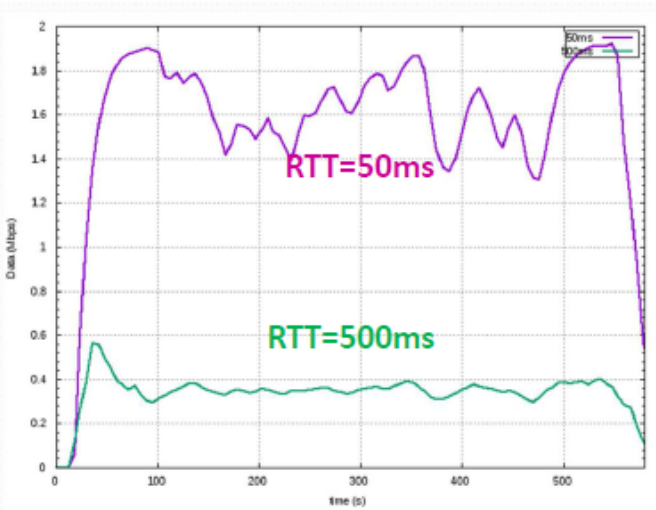
- Dynamic Adaptive Streaming over HTTPS (DASH):
  - Issue on clients in knowing the available bandwidth
  - Issues at server to reach the best available video playback quality
  - The client's requests could be adapted and specific traffic



Reno



BBR v1 (2018 results)



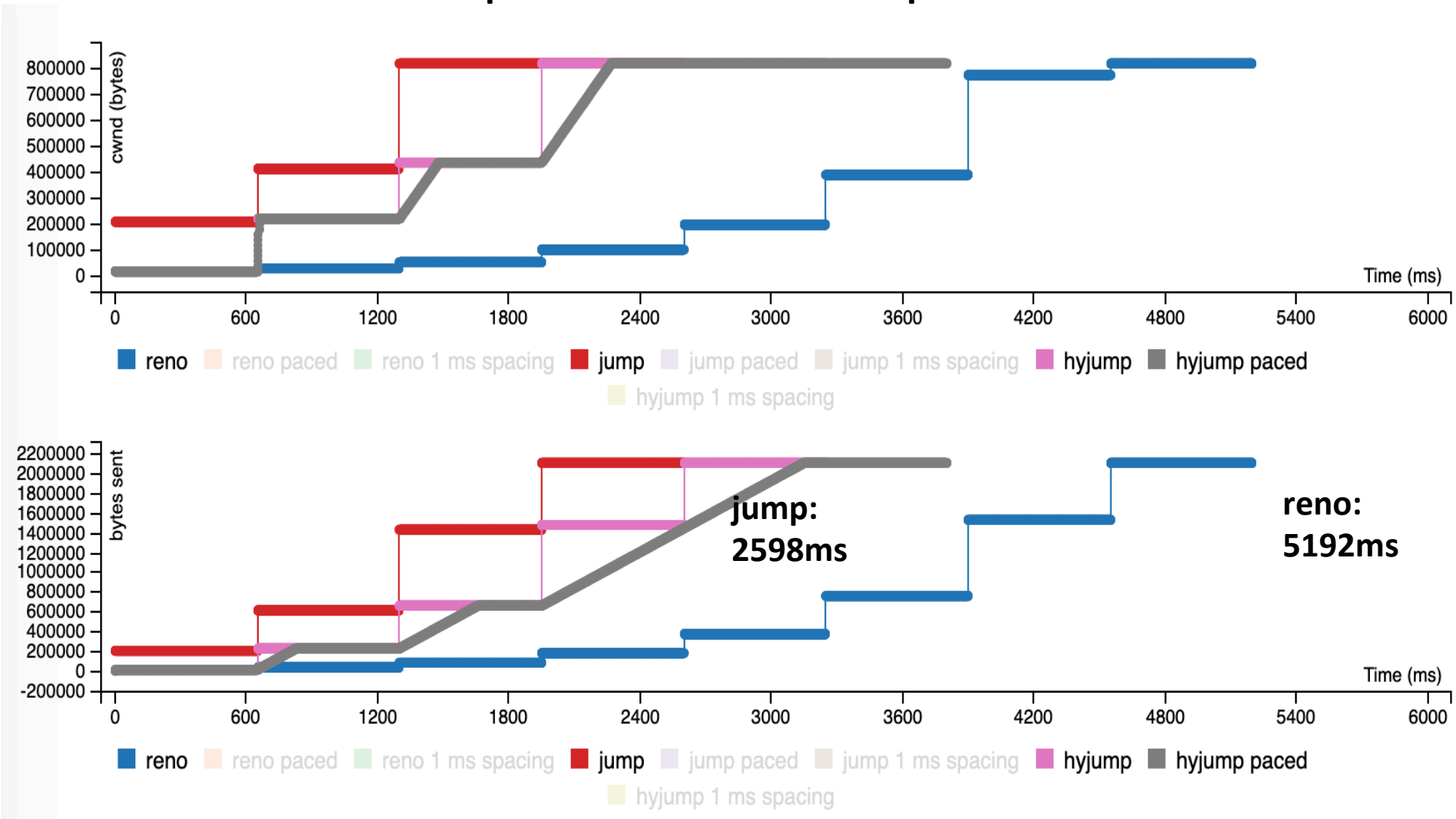
R. Secchi, D. Fernandes, R. Sallantin – presented at NetSat Days 2018

Motivation – A « safe » jump in cwnd

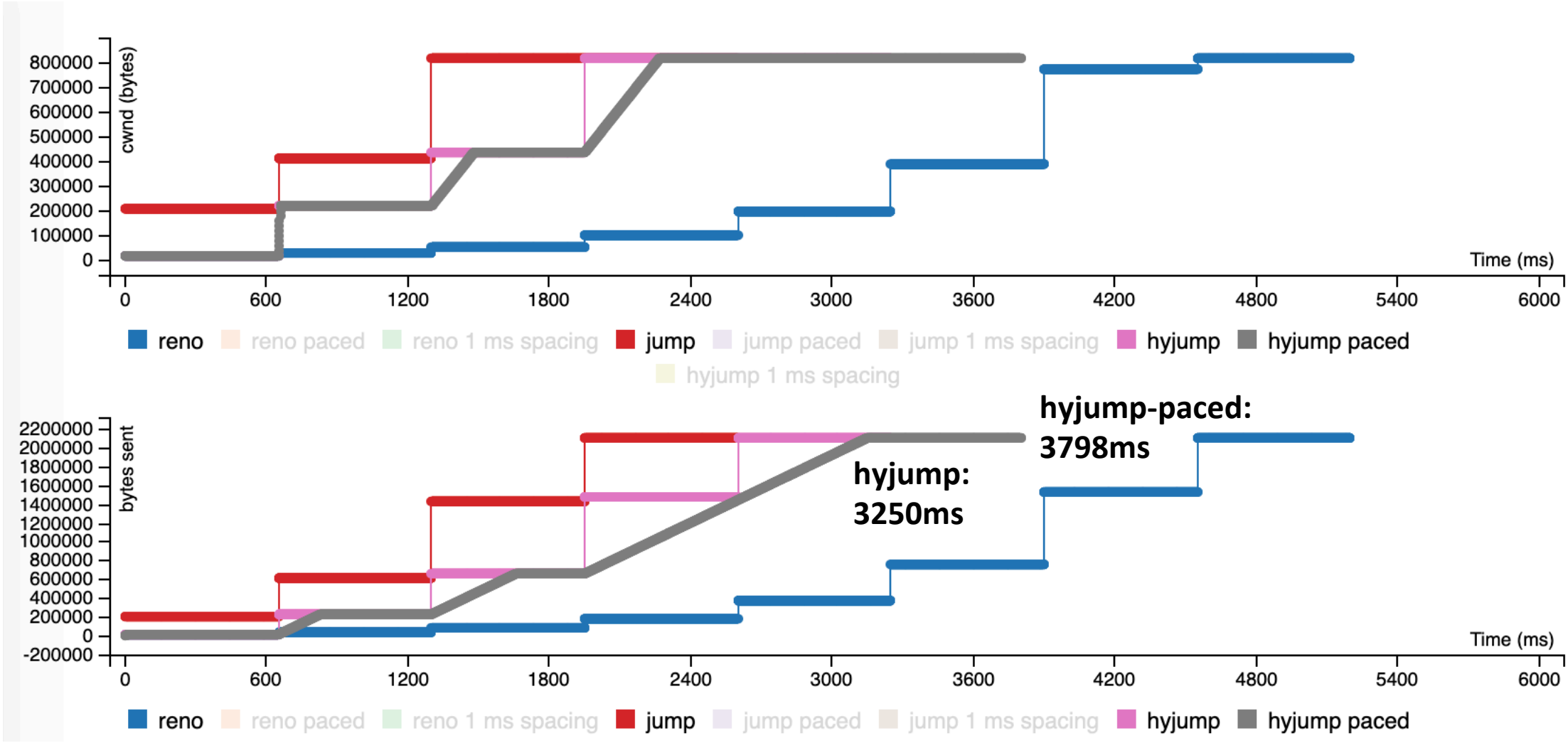
- Implementation of draft-kuhn-quic-0rtt-bdp-07:
  - Picoquic https://github.com/private-octopus/picoquic/pull/1073
  - Application level:2 MB transfer - median
- Network characteristics: draft-jones-tsvwg-transport-for-satellite
  - 50 Mbps download / 10 Mbps upload
  - RTT : 650 ms
- Congestion control
  - CUBIC
  - 0-RTT-BDP reaction:
    - jump to previously capacity (not recommended but “easy to implement” as a first step)
    - Beware the potential issue in using bytes_in_flight metric

Without 0-RTT	With 0-RTT	With 0-RTT-BDP
4,3 s	3,4 s	2,9 s

Reno and a Jump to 25% of previous cwnd



Hyjump: 25% jump ***after*** IW, then grow window to fill the capacity



Discussion - Client point-of-view

- Client can read the values of the extension
- Client may:
  - reject the extension (e.g. because connectivity changed)
  - accept and adapt the resource and flow control parameters
  - adapt application requests
- Client cannot change the values of the extension

Discussion - BDP extension protected as Much as initial_max_data

- For version 1 of QUIC:
  - BDP extension is protected using the mechanism that already protects the "initial_max_data" parameter
  - Defined in sections 4.5 to 4.7 of [I-D.ietf-quic-tls]
  - This allows the server to check parameters proposed by the client are those that the server sent to the client during the previous connection.

The need to sync between QUIC and TLS

- Proposed extension sits between TLS and QUIC specifications
- Which is not always clear :
  - see discussion in https://mailarchive.ietf.org/arch/msg/quic/7cSiXuuqGRjiRuKw7cfHreScuNc/
- There seems to be a difference between
  - Using QUIC as currently specified mapped with TLS1.3 implementation
  - Using QUIC and early_data without TLS1.3 implementation
  - Sync needed ?

Discussion – Congestion Control safety

- Some options to use the cwnd info:
  - Increase safely the initial congestion window [I-D.irtf-iccrg-sallantin-initial-spreading][CONEXT15]
    - Some CDN's currently exploit a very high Initial Window [TMA18]
  - Jump after IW (Hyjump – in the previous slides)

- We will need to back-out quickly when the jump is wrong!
- Do we need a draft on congestion safety ? (updating RFC 6928)

[TMA18] Ruth, J. and O. Hohlfeld, "Demystifying TCP Initial Window Configurations of Content Distribution Networks", 2018 Network Traffic Measurement and Analysis Conference (TMA) , 2018.

[CONEXT15] Li, Q., Dong, M., and P. Godfrey, "Halfback: Running Short Flows Quickly and Safely", ACM CoNEXT , 2015.

[I-D.irtf-iccrg-sallantin-initial-spreading] Sallantin, R., Baudoin, C., Arnal, F., Dubois, E., Chaput, E., and A. Beylot, "Safe increase of the TCP's Initial Window Using Initial Spreading", draft-irtf-iccrg-sallantin-initial-spreading-00 (work in progress), January 2014.

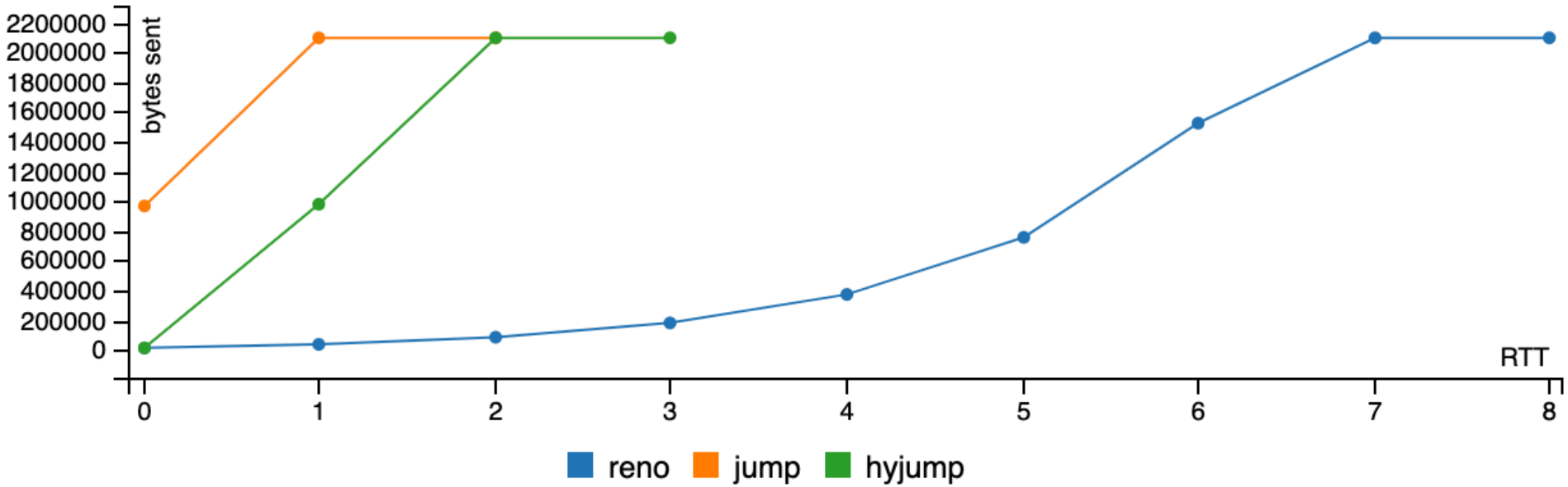
Next Steps

- Use a newBDP extension in QUIC ?
  - Or
- Update the standard behavior ?
  - “For 0-RTT data to be sent, the QUIC server must record the values of:
    - initial_max_data
    - initial_max_stream_data_bidi_local
    - initial_max_stream_data_bidi_remote
    - initial_max_stream_data_uni
    - initial_max_streams_bidi
    - initial_max_streams_uni”
    - recon_bytes_in_flight
    - recon_min_rtt

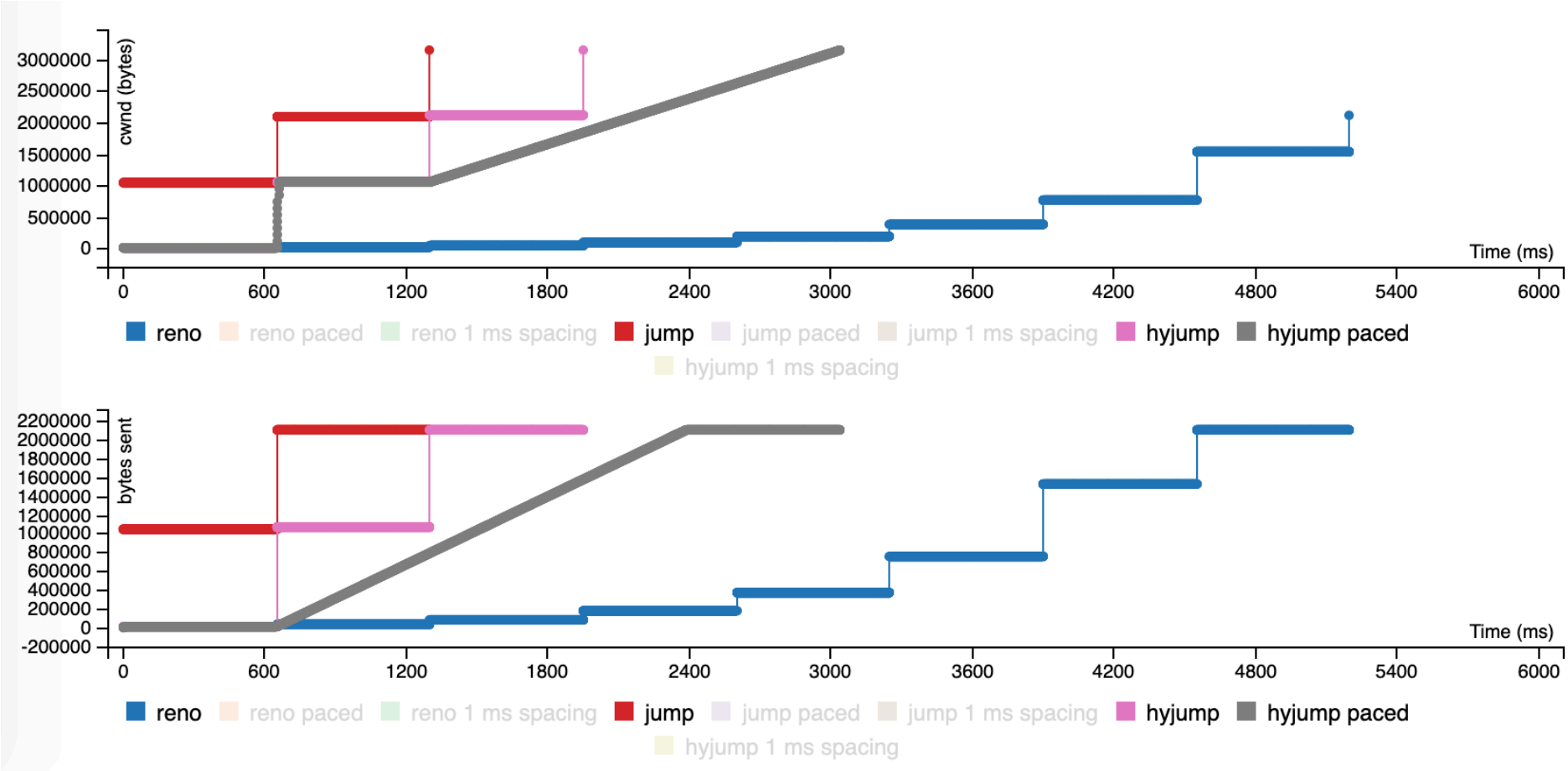
Additional Slides

Simple model for 50/10 Mbps

As capacity increases the effects become greater!



Hybrid: 25% jump after IW, then grow window to fill capacity for 50/10 Mbps



draft-kuhn-quic-0rtt-bdp params:

o recon_bytes_in_flight (0x000X): The bytes in flight measured on the previous connection by the server. Integer number of bytes. Using the bytes_in_flight defined in [I-D.ietf-quic-recovery], recon_bytes_in_flight can be set to bytes_in_flight.

o recon_min_rtt (0x000X): The minimum RTT measured on the previous connection by the server. Integer number of milliseconds. Using the min_rtt defined in [I-D.ietf-quic-recovery], recon_min_rtt can be set to min_rtt. The min_rtt parameter may not track a decreasing RTT: the min_rtt that is reported here may not be the actual minimum RTT measured during the 1-RTT connection, but still reflects the characteristics of the latency on the network.