Understanding Prague for L4S

Including latest updates

draft-briscoe-iccrg-prague-congestion-control

Many contributors in Open-Source repositories:

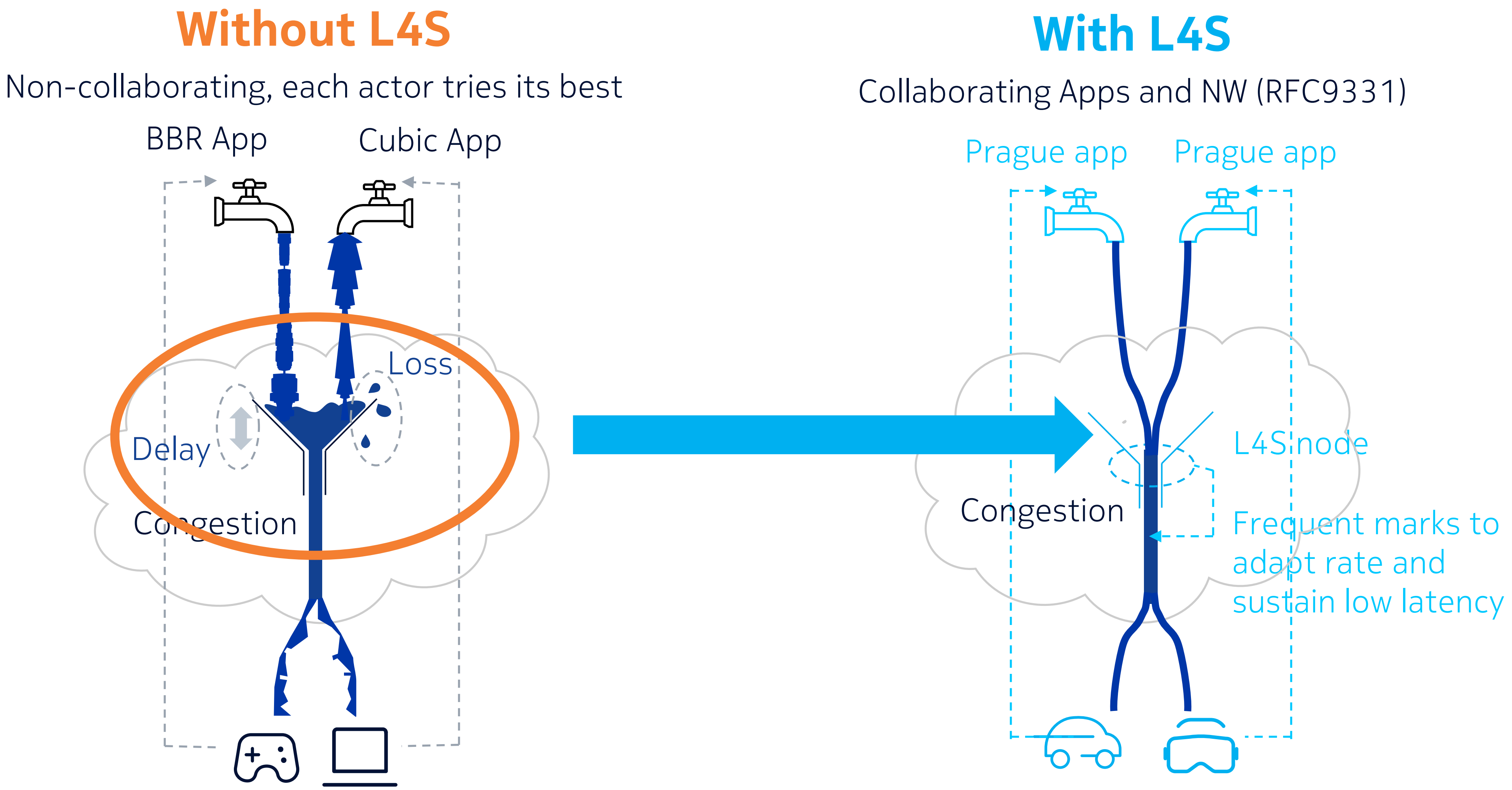
L4STeam/linux: Kernel tree with TCP-Prague and DualPI2

L4STeam/udp_prague: UDP-Prague CC object and examples (still under construction)

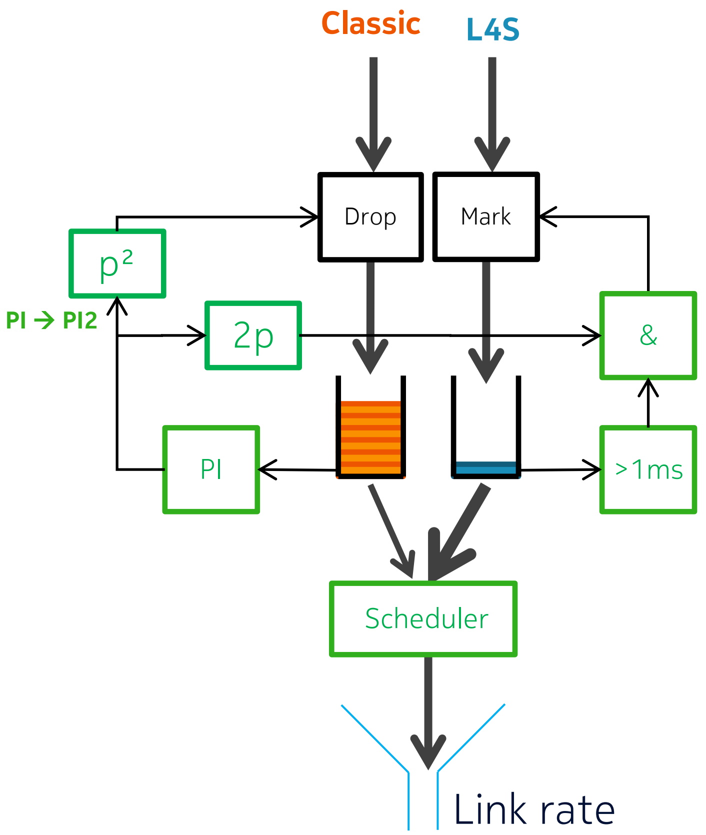
Presenter: Koen De Schepper

(koen.de_schepper@nokia-bell-labs.com)

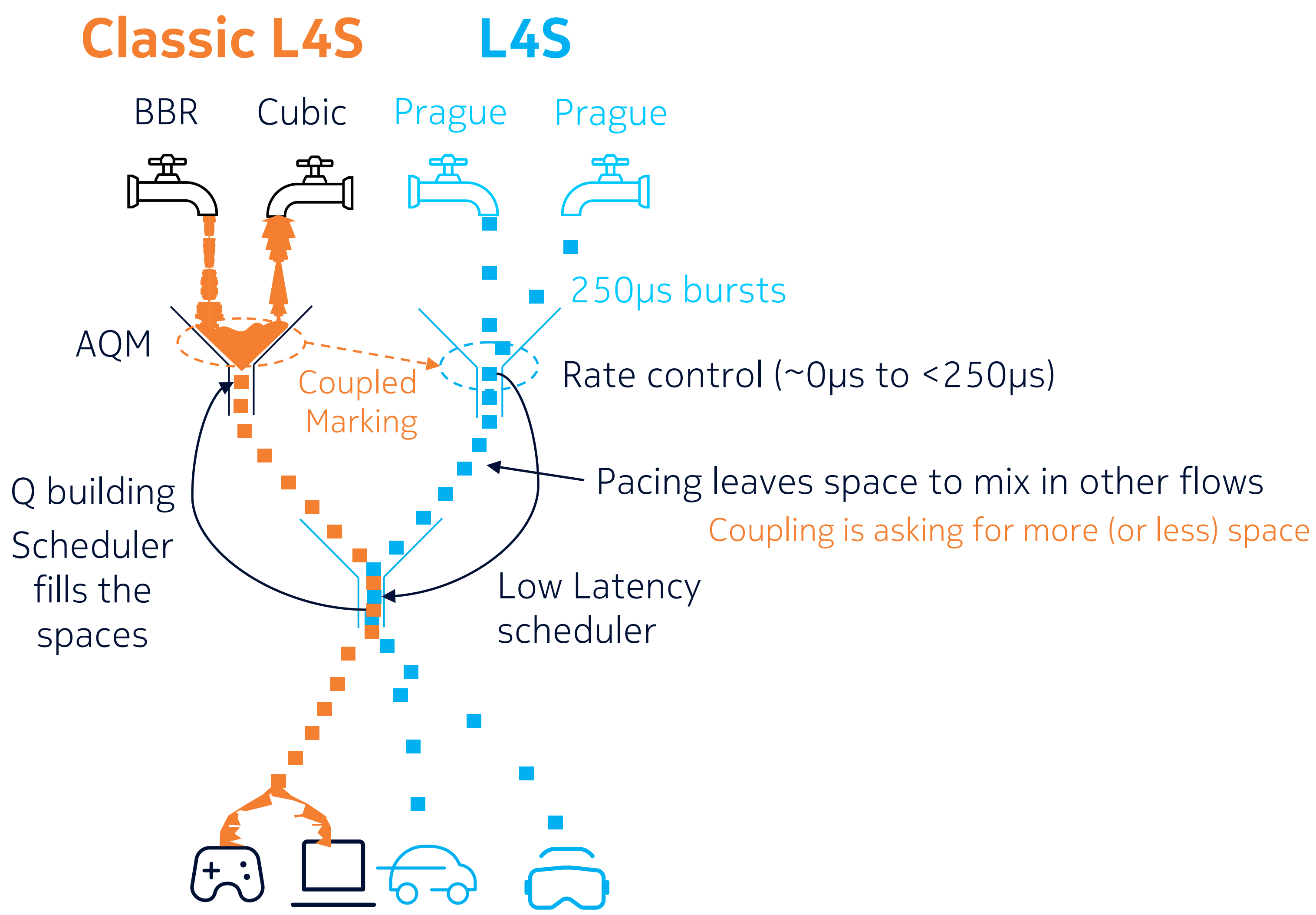
Why Classic NEEDS a buffer and how it is avoided with Prague



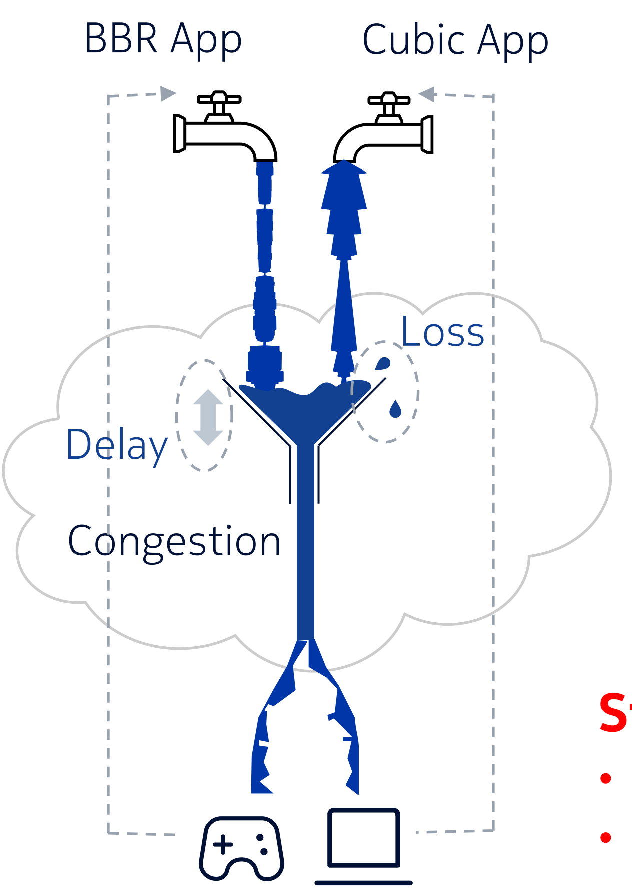
Why Prague doesn't build a Queue

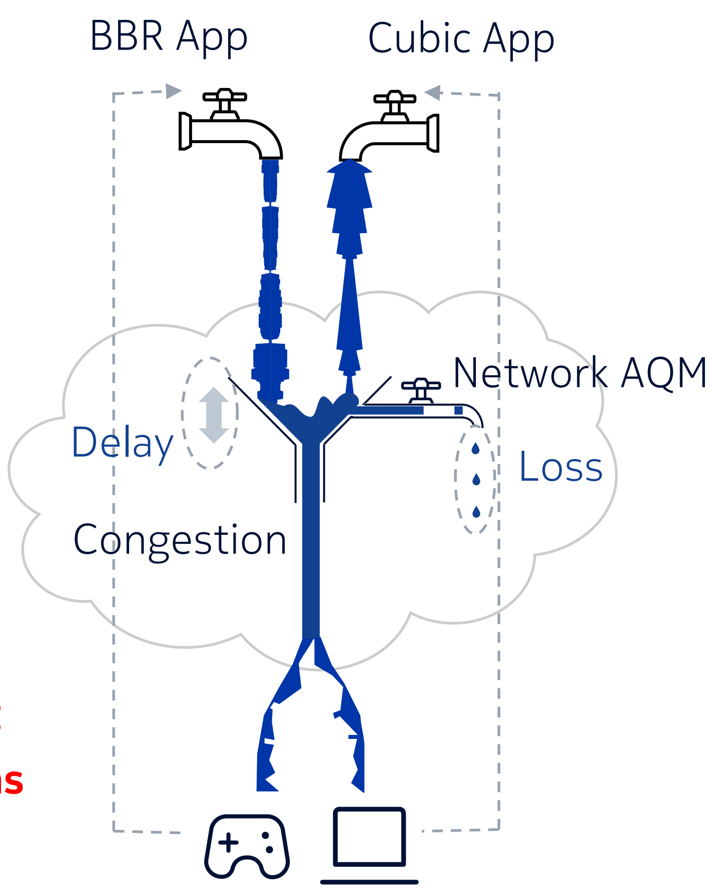


DualPI2 (RFC9332)



Lower Latency: Drop with AQM



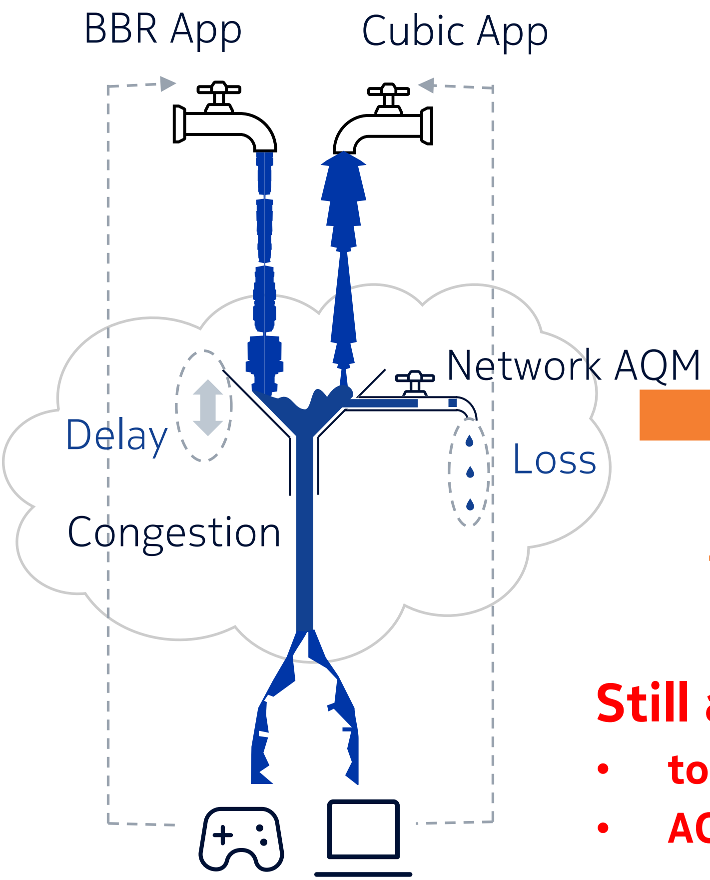


Still a queue needed:

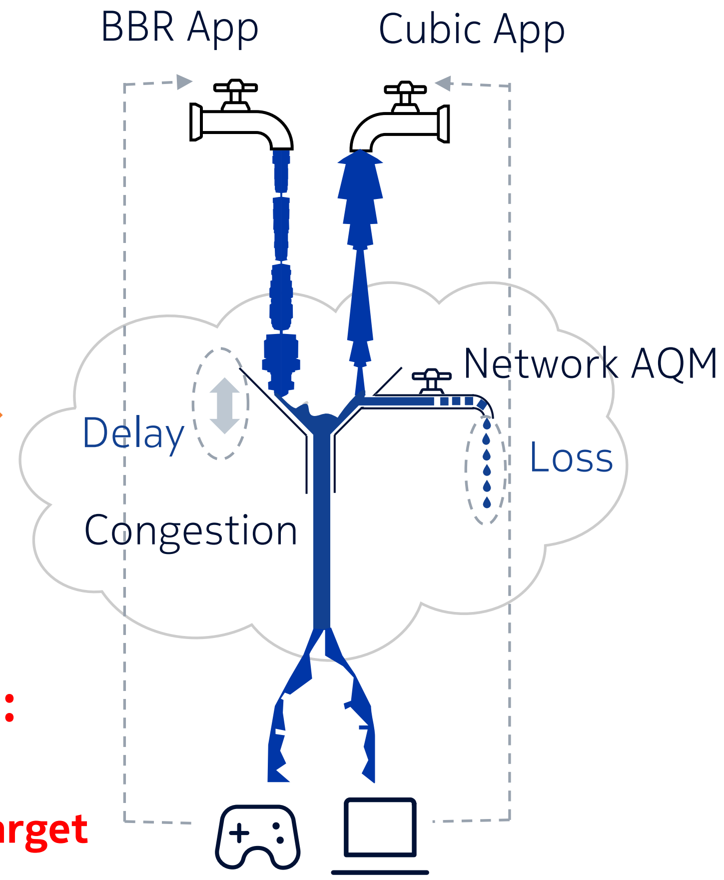
- **to cover rate variations**
- **to control the rate of delay based CCs**

Lower Latency = Higher Loss

AQM with higher target



AQM with lower target



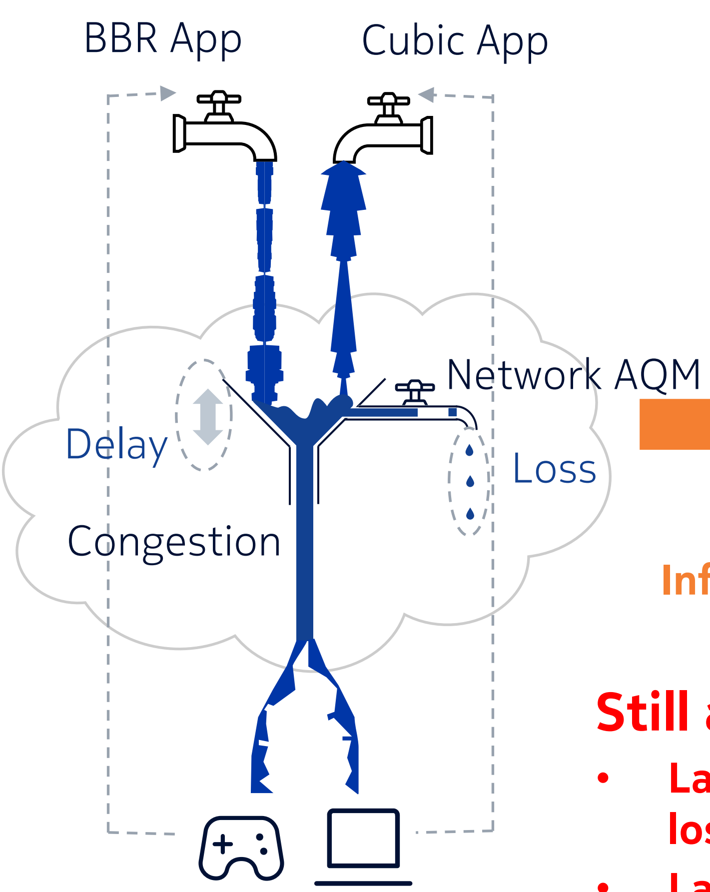
Too low target, too high drop

Still a queue needed:

- **to limit loss rate**
- **AQM target <> BBR target**

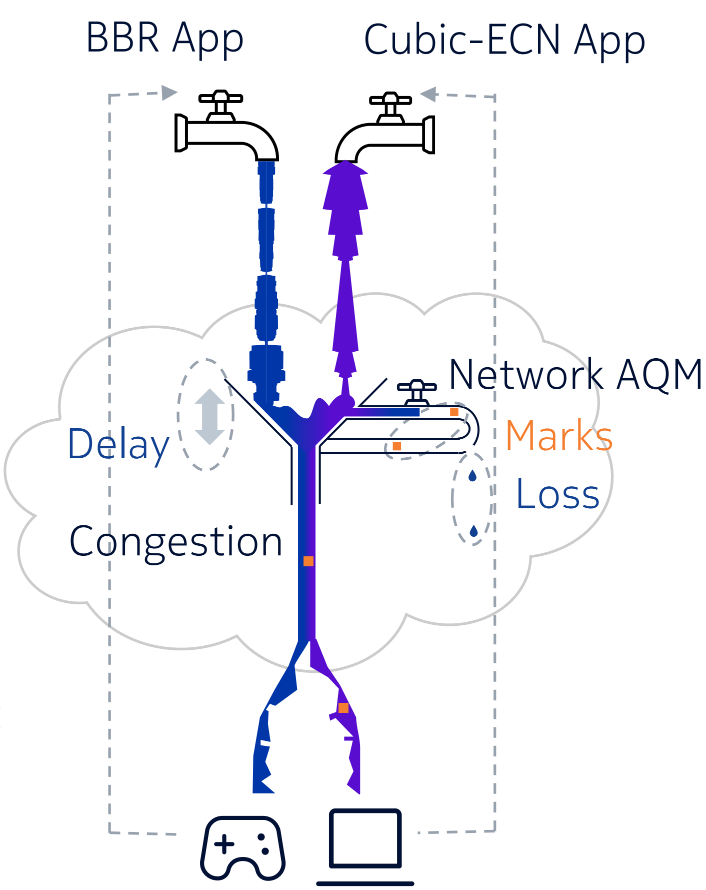
Lower Loss: Mark with ECN instead of drop

Without ECN



With ECN

(old RFC3168)



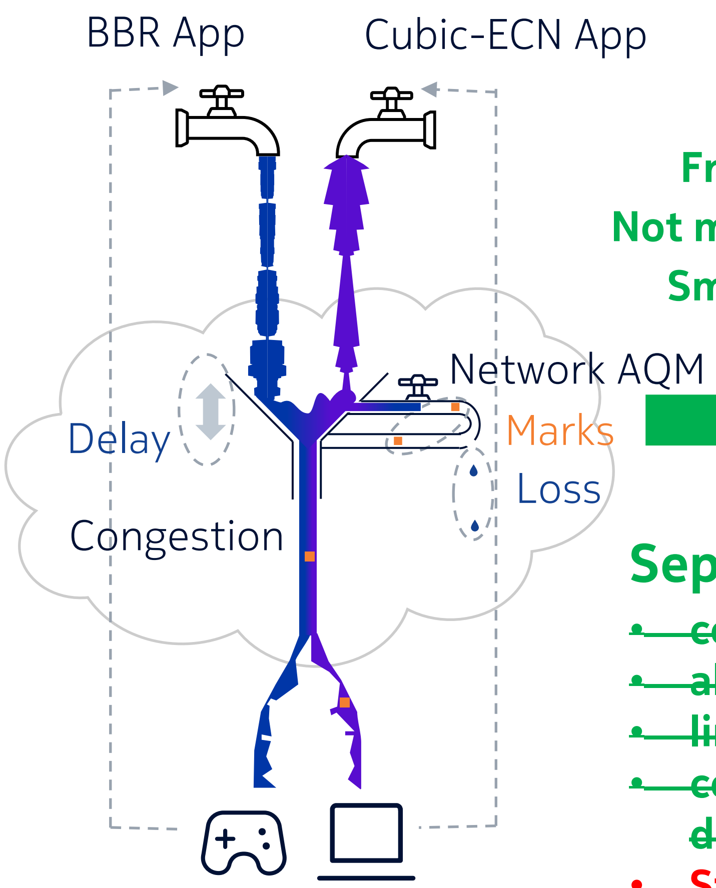
Infrequent marking, same as drop

Still a queue needed:

- **Large enough for loss-based**
- **Large enough for BBR delay-based**

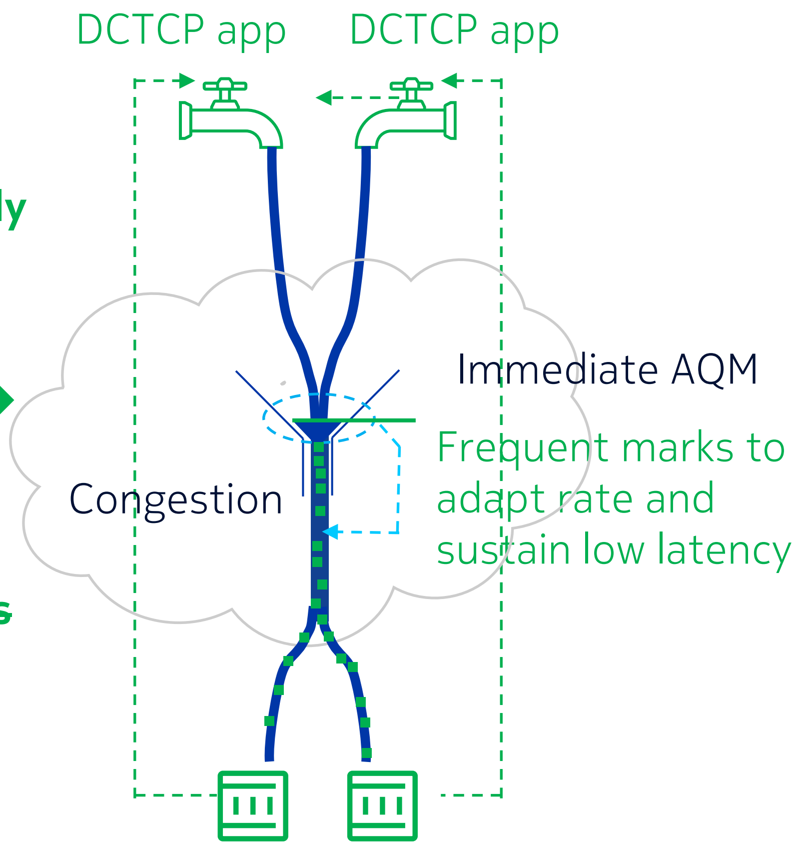
Lower Latency: Mark frequent with DCTCP

With ECN



Frequent marking,
Not mixed with drop-only
Smooth throughput

With DCTCP ECN

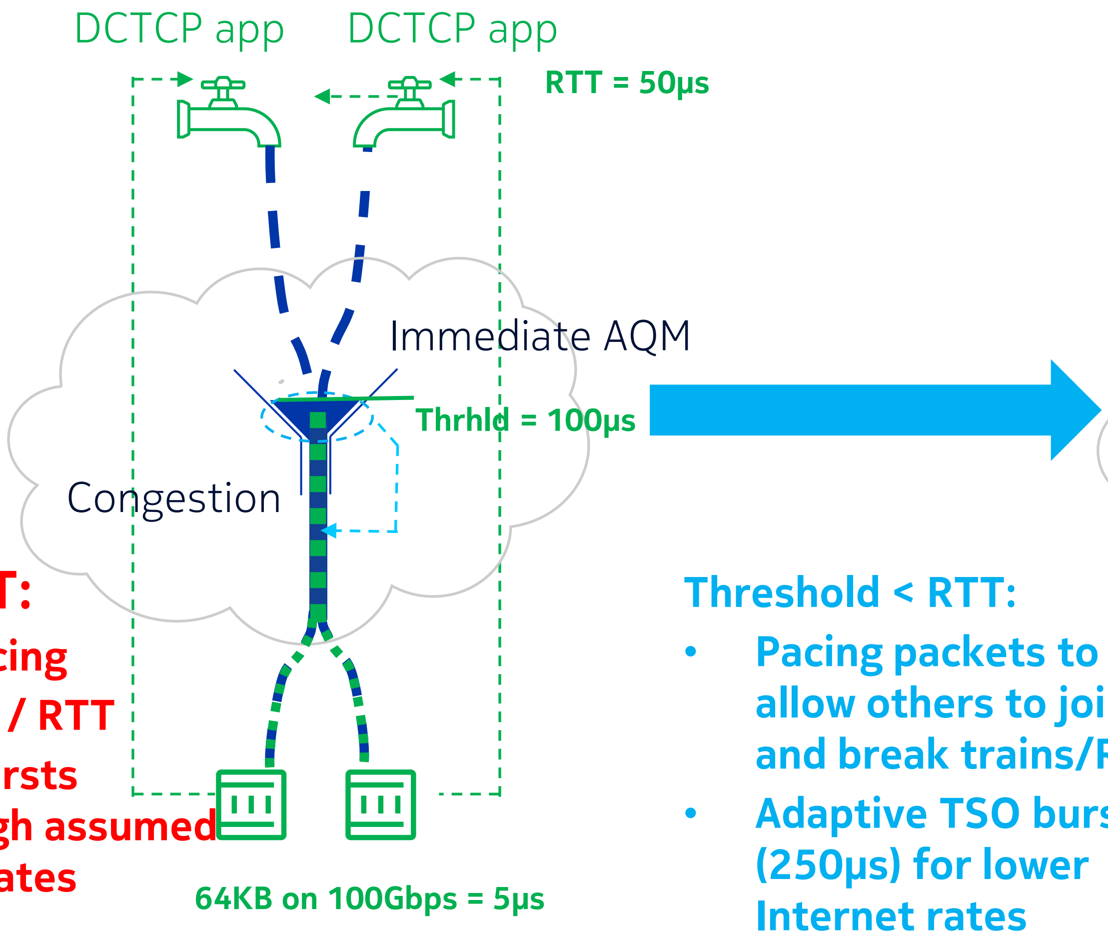


Separate network:

- ~~cover rate variations~~
- ~~all support ECN~~
- ~~limit loss rate~~
- ~~control the rate of delay-based CCs~~
- **Still queues needed: order > base-RTT**

Lower Burst: Pacing and Scaling TSO size with the sending rate

With DCTCP ECN

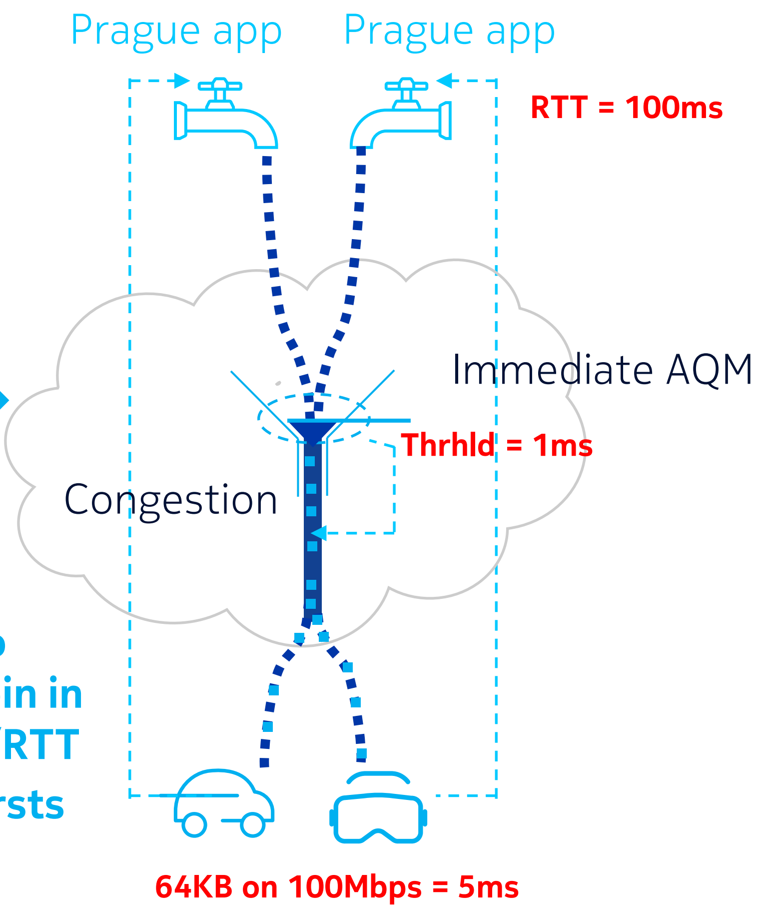


Threshold > RTT:

- **Only ACK-Pacing**
- **Packet trains / RTT**
- **Large TSO bursts (64KB) for high assumed DataCenter rates**

With L4S

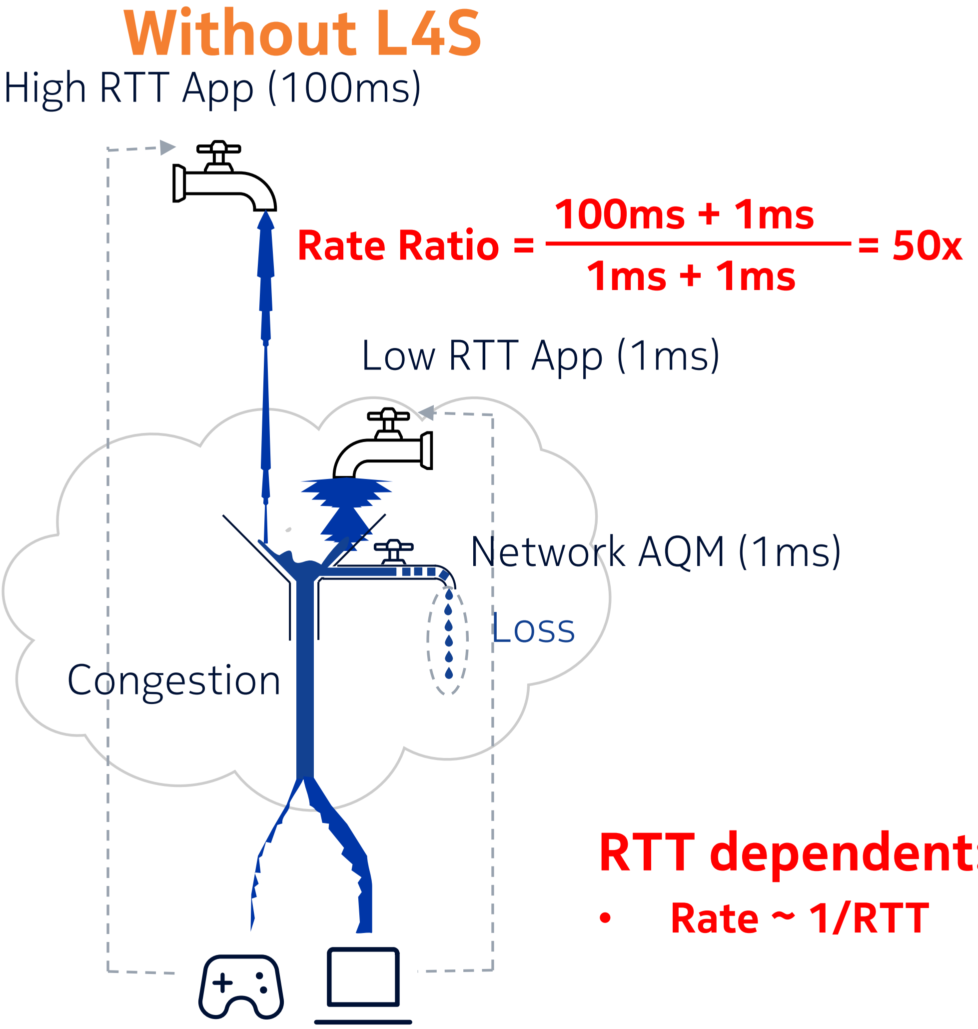
Collaborating Apps and NW (RFC9331)

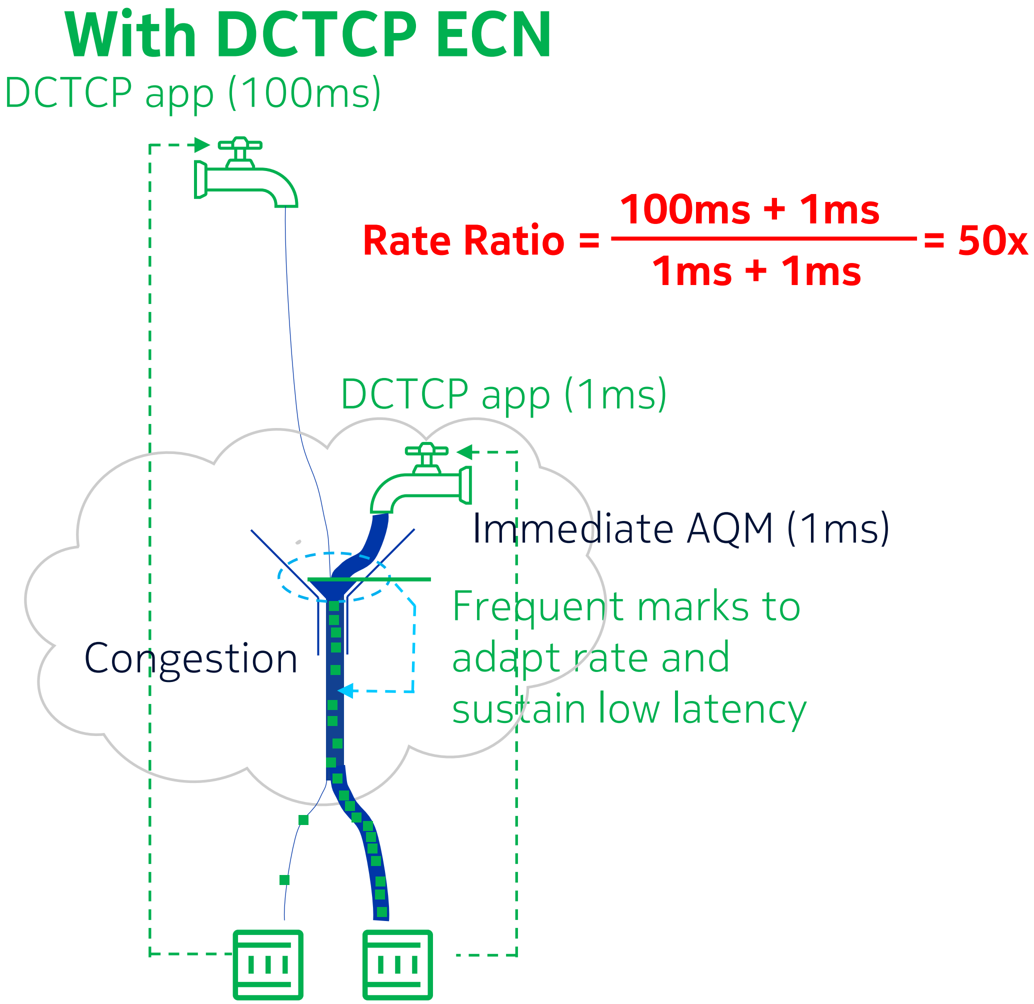


Threshold < RTT:

- **Pacing packets to allow others to join in and break trains/RTT**
- **Adaptive TSO bursts (250µs) for lower Internet rates**

RTT unfairness: Very Low RTT = very High Rate



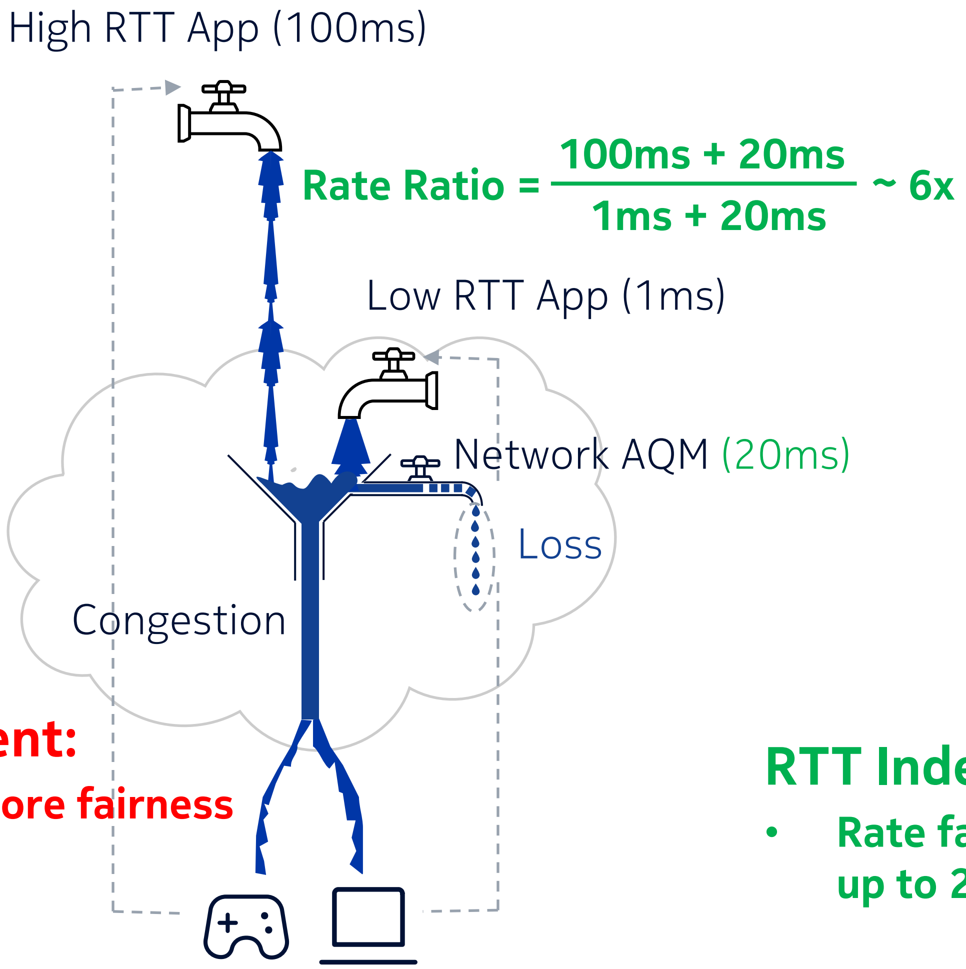


RTT dependent:

- **Rate ~ 1/RTT**

RTT unfairness: From NW ➔ Prague responsibility

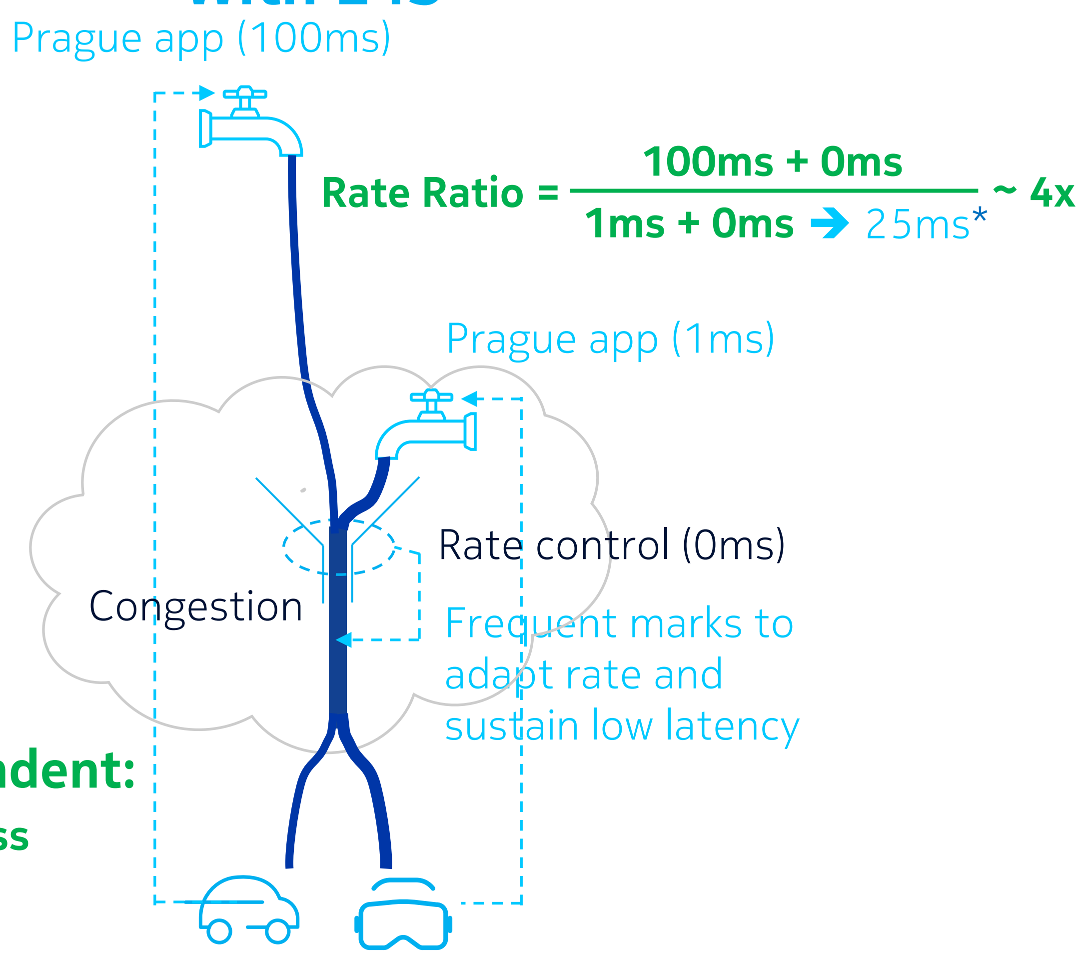
Without L4S



RTT dependent:

- **Bigger Q= more fairness**

With L4S



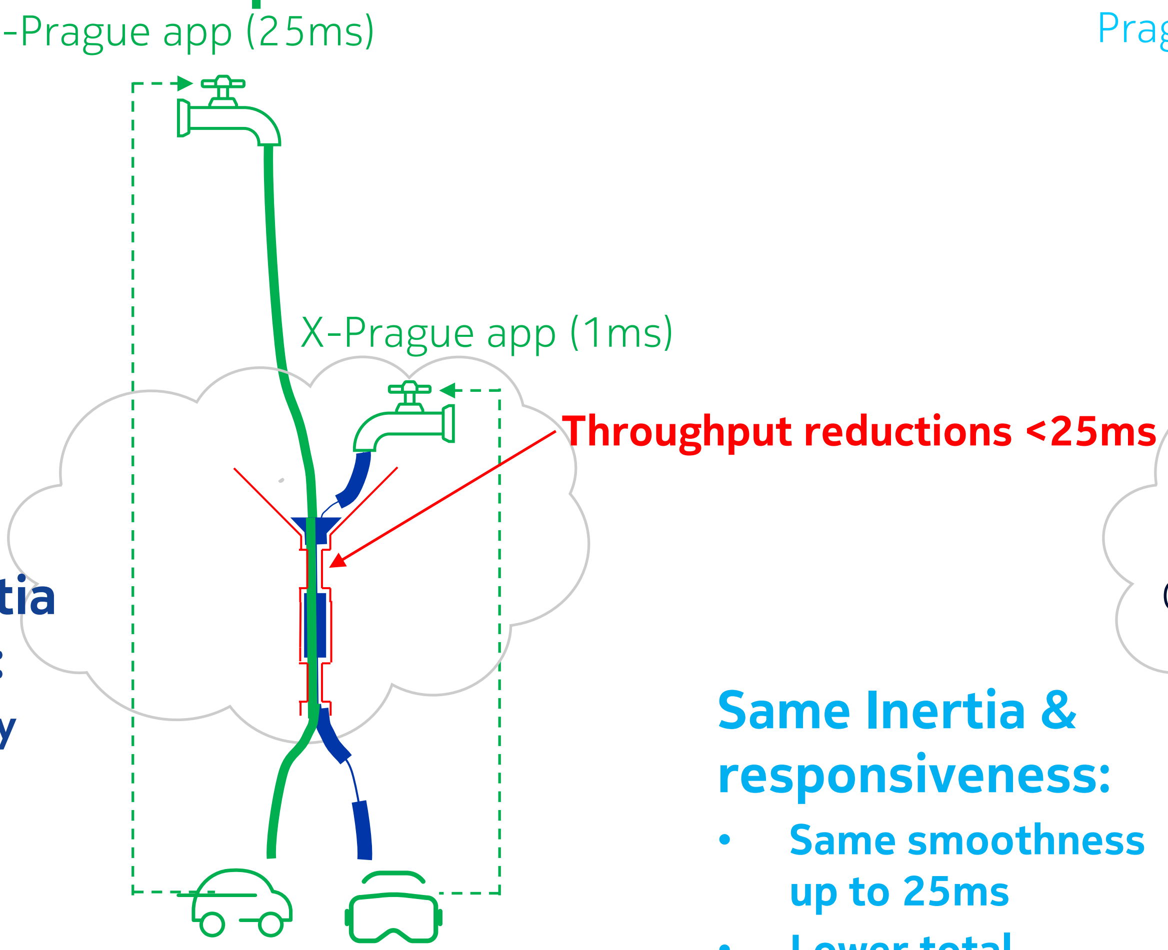
RTT Independent:

- **Rate fairness up to 25ms**

*Note: The app experienced RTT stays 1ms, only how it responds to rate changes is like 25ms

RTT unfairness: Responsiveness ↔ Inertia

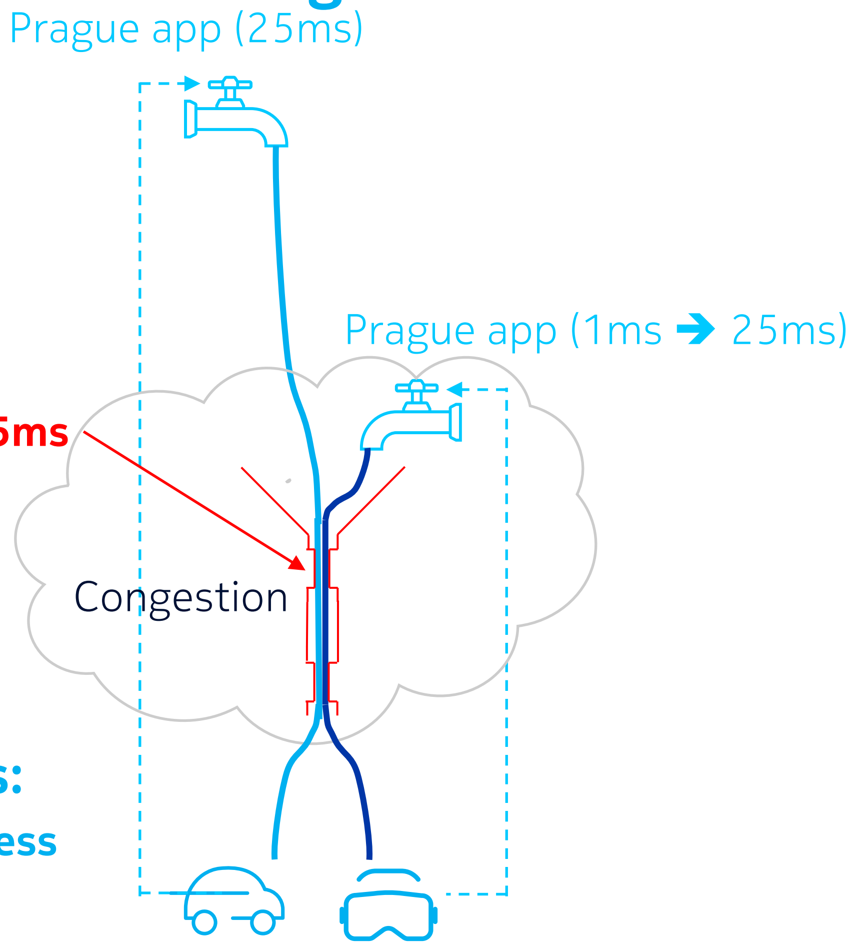
With RTT-dependent Inertia



Lowest Inertia gets bullied:

- Large highly frequent variations

With Prague L4S



Same Inertia & responsiveness:

- Same smoothness up to 25ms
- Lower total throughput

New: Pacing below Minimum window: Minimum rate = f(RTT, Nbr)

Minimum window = 2 packets

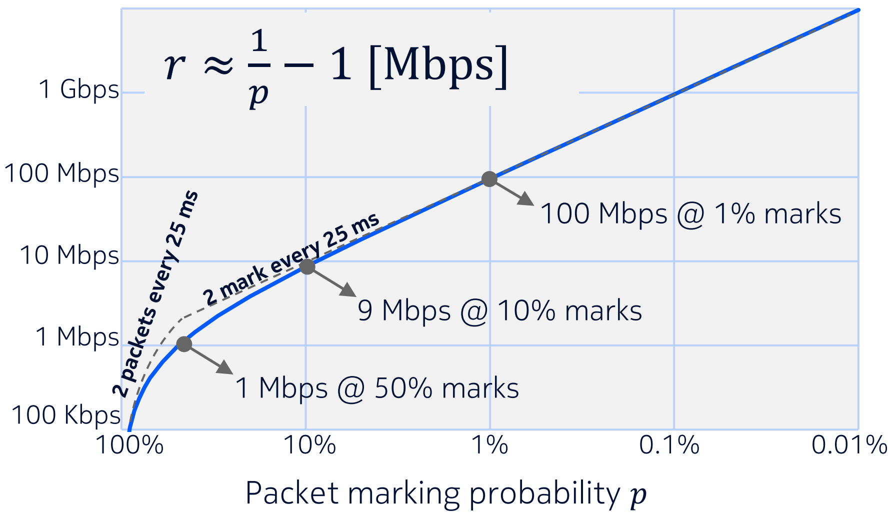
High Rates possible
Not controllable under this rate with marks

Window [packets]	Nbr of Flows	Rtt [ms]	Old Rate [Mbps]	New Rate [Mbps]
2	1	10	2.4	0.1
2	10	10	24	1
2	100	10	240	10
2	1	1	24	0.1
2	10	1	240	1
2	100	1	2400	10
2	1	0.1	240	0.1
2	10	0.1	2400	1
2	100	0.1	24000	10

Rates go down to 100kbps per flow

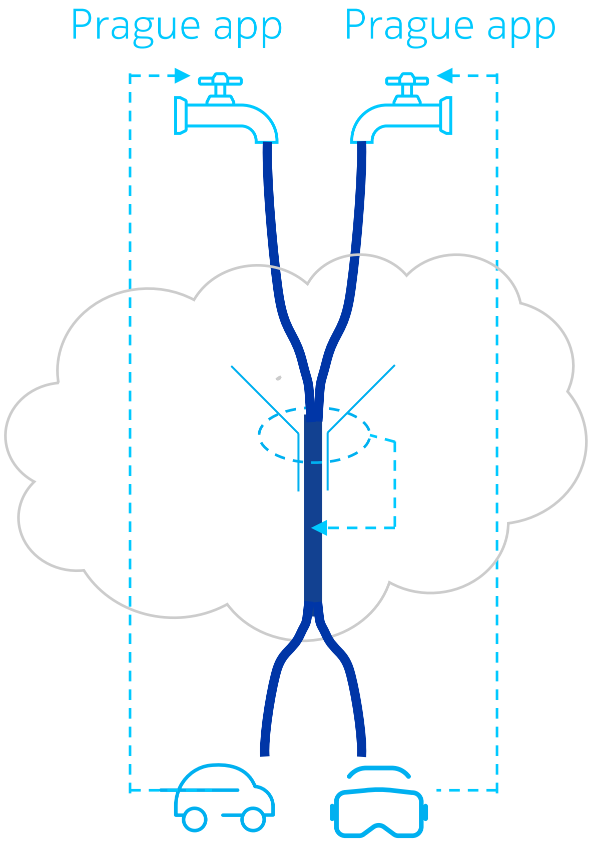
Allowing more flows in lower BW independent of RTT

Minimum window: Minimum rate = f(RTT)



With L4S

Collaborating Apps and NW (RFC9331)



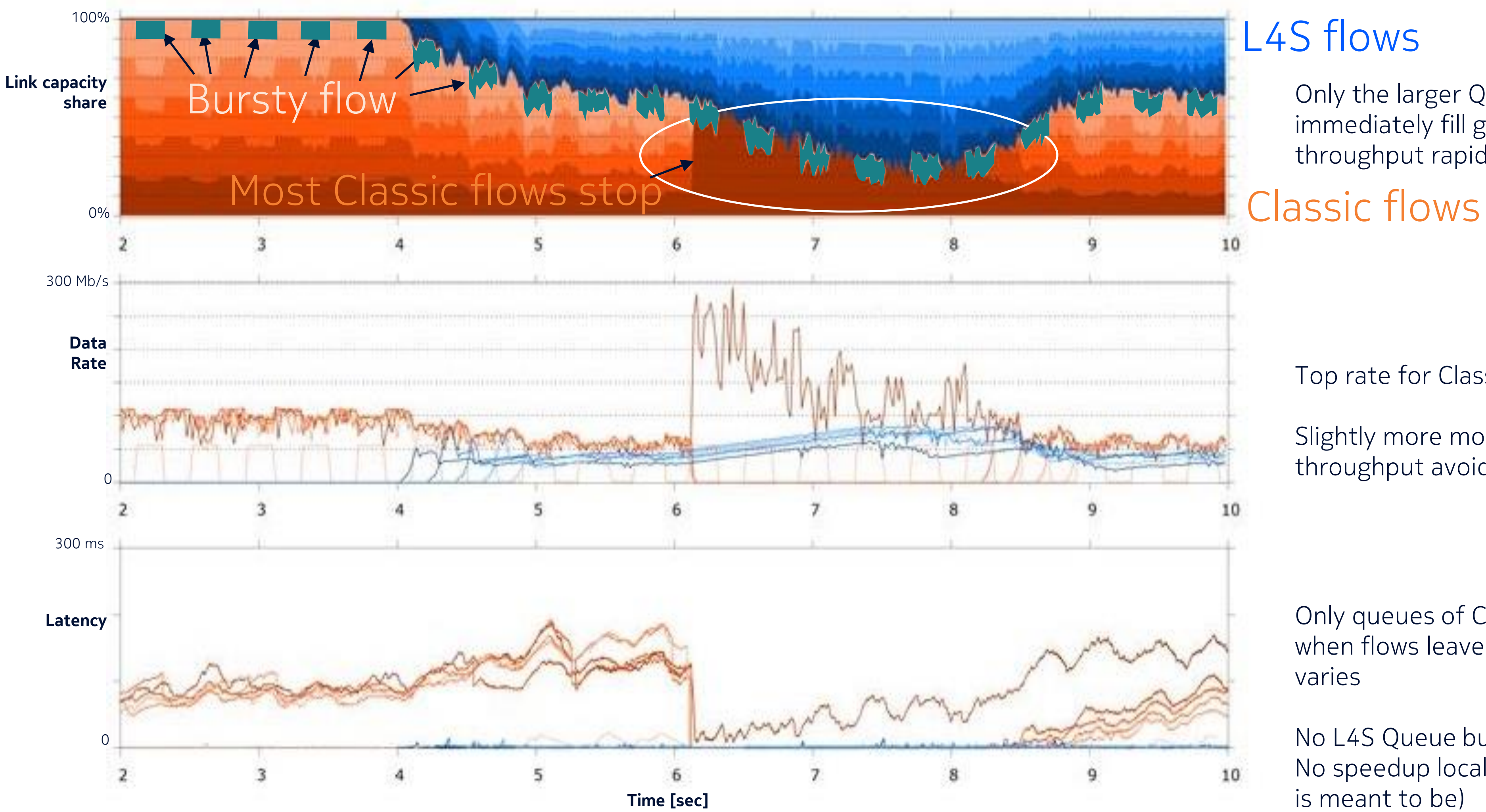
Newest Prague (ported from RT-Prague):

- Update Fractional Window from ECE → smaller steps for Pacing rate
  → roundup as the integer packet TCP stack Window, min 2 packets
  → below 1Mbps, packet size is reduced to keep 2 packets per 25ms

- Minimum Rate of 100kbps, maximum packet send latency 12.5ms

- Prague code: L4STeam/linux: Kernel tree containing patches for TCP Prague and the dualpi2 qdisc (github.com)

Still compromizes: Responsiveness, Rate, Latency, Smoothness



Only the larger Queues of Classic traffic can immediately fill gaps when flows leave or if throughput rapidly varies

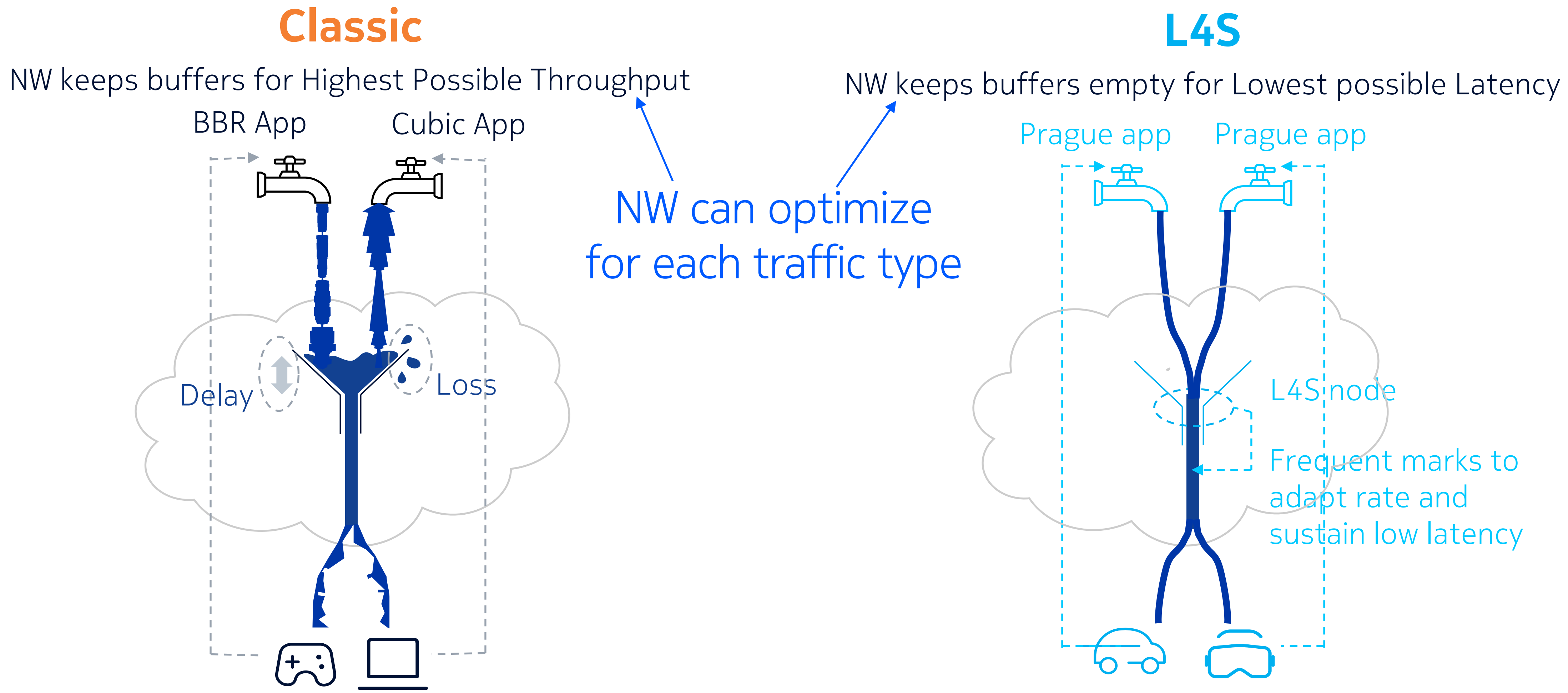
Top rate for Classic when opportunities arise

Slightly more moderate, but smoother L4S throughput avoids latency spikes

Only queues of Classic traffic can fill gaps when flows leave or if throughput rapidly varies

No L4S Queue buildup
No speedup local data backup (is what a queue is meant to be)

With L4S: Apps can still choose between 2 types of traffic
No need for the NW to compromise in the middle



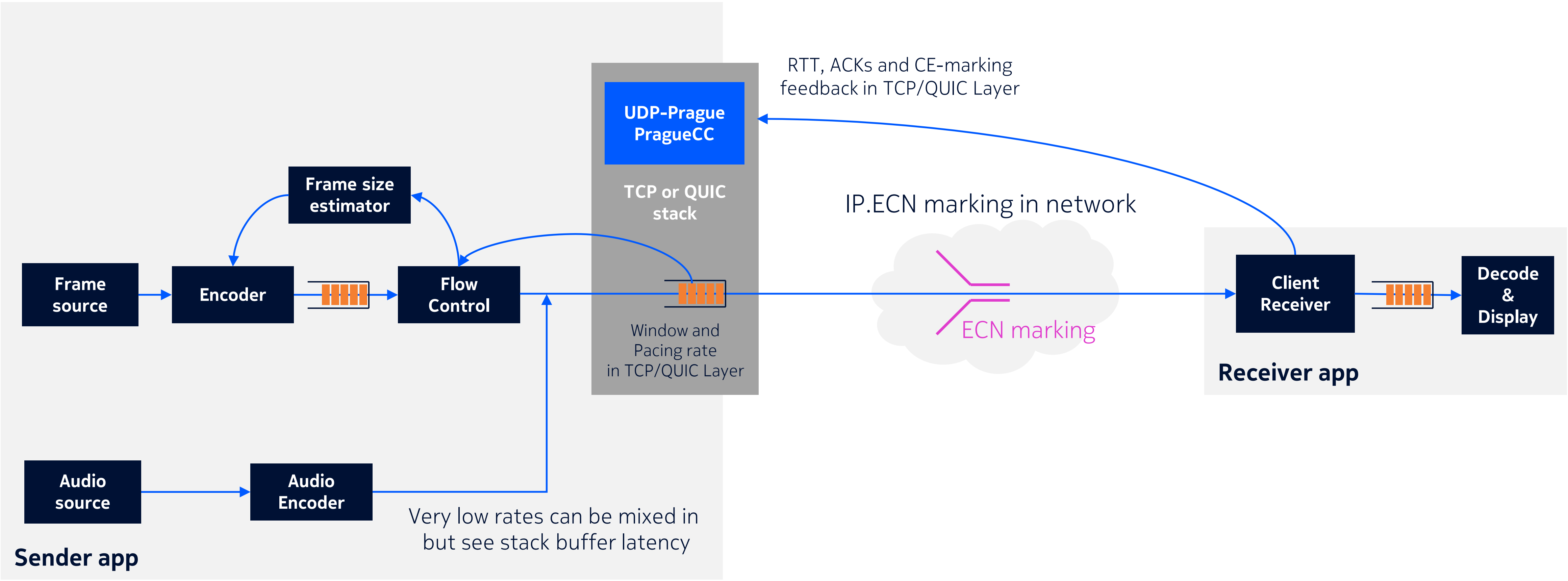
UDP-Prague

GitHub repository: L4STeam/udp_prague

- Prague congestion control protocol for UDP-based applications targeting very interactive user experience
- Single source C++ PragueCC object with minimal dependencies
- Platform independent code for PragueCC
- Allows evolution of Prague without impacting the API (so regularly check for now)
- Allows directly controlling application data generation rate and non-blocking delivery (API can still evolve)
- Examples present for sender and receiver (can work bidirectionally)
- Iperf2 integration ongoing at: Iperf 2 / Code / udp_prague
- Under construction, Todo:
  - PRR-like reduction,
  - app limit window growth limit,
  - RT-Prague API for Video,
  - Cubic on loss and/or faster capacity search…
  - More examples
  - …
- Let us know if you are using this and have suggestions on API or CC improvements

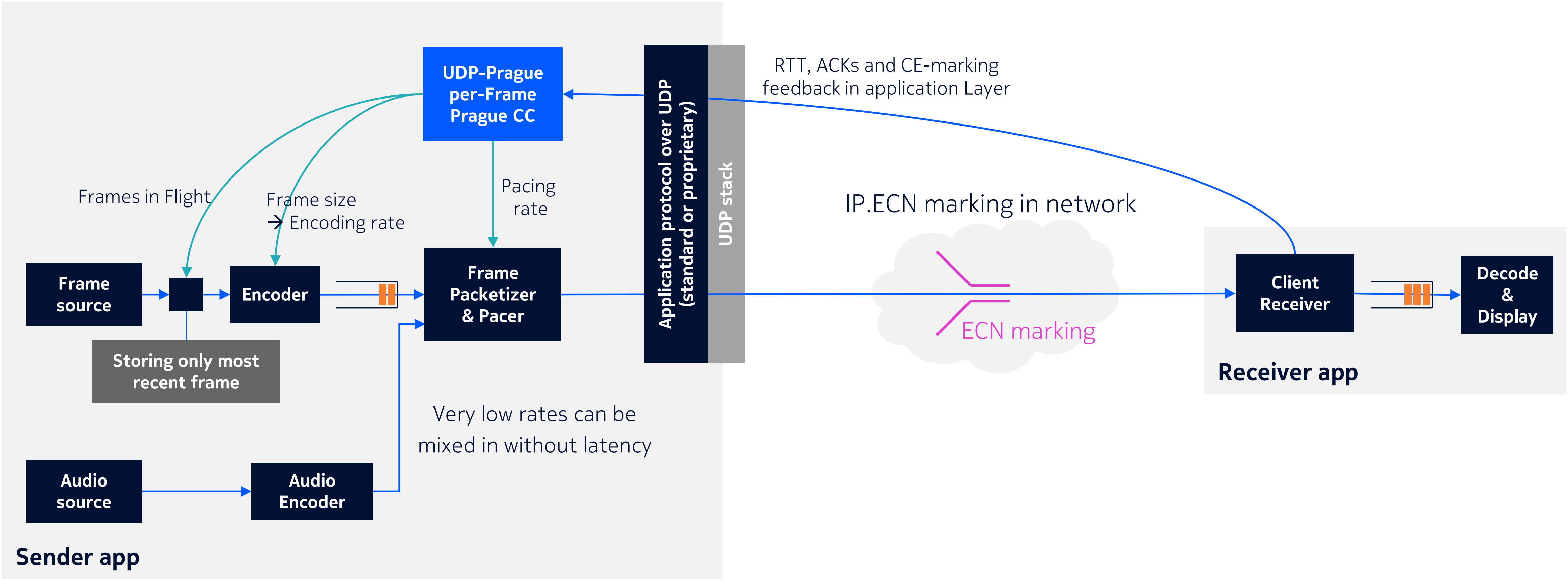
UDP-Prague with TCP or QUIC stacks

Stacks can use PragueCC object



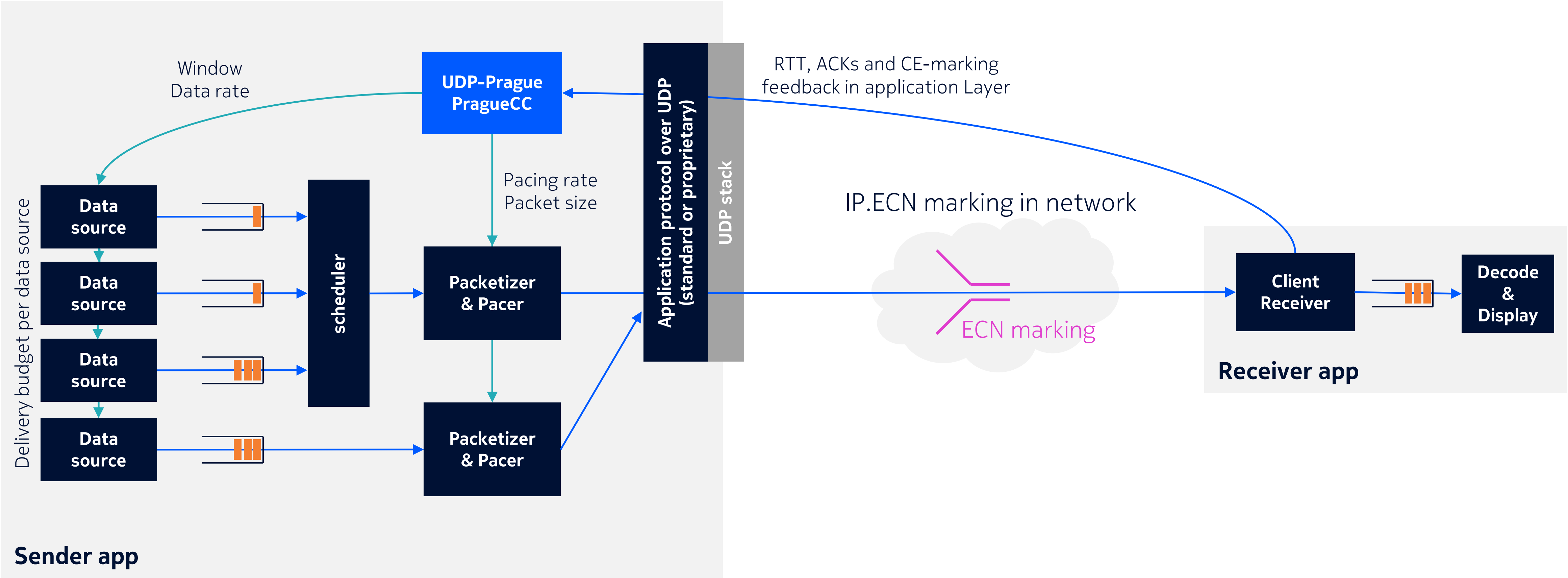
Real-Time app optimization with UDP/RT-Prague

PragueCC uses ECN feedback directly to determine the next frame and its size



Real-Time app optimization with UDP-Prague

PragueCC uses ECN feedback directly to determine the pacing rate and window



Questions?

draft-briscoe-iccrg-prague-congestion-control

Many contributors in Open-Source repositories:

L4STeam/linux: Kernel tree with TCP-Prague and DualPI2

L4STeam/udp_prague: UDP-Prague CC object and examples (still under construction)

Presenter: Koen De Schepper

(koen.de_schepper@nokia-bell-labs.com)

Prague Status

- Apple QUIC-Prague
  - Falls back to Cubic on loss
  - Beta in MacOS13 and iOS16, Released in MacOS14 and iOS17

- Linux TCP-Prague (recent features explained in ICCRG meeting on Friday)
  - For kernel versions:
    5.15 L4STeam/linux, 6.1 minuscat/l4steam6.1.y, 6.6 minuscat/l4steam-6.6.y, 6.7 minuscat/net-next/tree/upstream_l4steam
  - Rpi 6.6 minuscat/rpi-6.6.y
  - 6.11 (for main lining soon)

- UDP-Prague L4STeam/udp_prague
  - Prague congestion control protocol for UDP-based applications targeting very interactive user experience
  - Single source C++ reference PragueCC object directly controlling application data generation rate
  - Further additions: RT-Prague for Video, Cubic on loss, ...

- Iperf2 Iperf 2 / Code / udp_prague
  - Supporting precise app level latency measurements (without socket buffering)

- Prague based applications over UDP (like Nvidia GeForce Now)

- Draft exists in ICCRG: draft-briscoe-iccrg-prague-congestion-control

Recap

A Classic Queue is needed to:

- **cover rate variations**
- **control the rate of delay-based CCs**
- **limit loss rate**
- **hold large TSO bursts**
- **hold ACK-Pacing packet train bursts**
- **resolve RTT unfairness**
- **hold minimum window of 2 packets per flow**

- **Also, if Classic ECN is used: Above reasons stay as loss-based traffic is in the same Queue**

Classic can fill gaps in link utilization because of bigger Q

Better for data transfers that take > 1s

Latest L4S implementation:

- **can be controlled down to 100kbps per flow**
- **is RTT independent down to an RTT of 0µs and up to 25ms**
- **settles at 2 marks (1 at 1Mbps) per 25ms (or larger RTT)**
- **the marking intensity is higher if less packets are send per RTT**
- **sends always at least 2 packets per 25ms (or larger RTT to allow 2 marks)**
- **sends a packet per 10ms (with smaller MTU when rate <1Mbps)**

L4S can run without a Queue in the NW

Better for transactions that take << 1s