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

**Without L4S**
Non-collaborating, each actor tries its best
- BBR App
- Cubic App

**With L4S**
Collaborating Apps and NW (RFC9331)
- Prague app
- Prague app

Frequent marks to adapt rate and sustain low latency
Why Prague doesn’t build a Queue

DualPI2 (RFC9332)
Lower Latency: Drop with AQM

Without AQM

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

BBR App  Cubic App  Network AQM
Delay  Congestion  Loss

AQM with lower target

BBR App  Cubic App  Network AQM
Delay  Congestion  Loss

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**

- BBR App
- Cubic App

**With ECN** (old RFC3168)

- BBR App
- Cubic-ECN App

**Network AQM**

- **Delay**
- **Congestion**
- **Loss**

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

- BBR App
- Cubic-ECN App
- Delay
- Congestion
- Network AQM
- Marks
- Loss
- Frequent marking, Not mixed with drop-only Smooth throughput

With DCTCP ECN

- DCTCP app
- Frequent marks to adapt rate and sustain low latency

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**

DCTCP app

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

**With L4S**

Collaborating Apps and NW (RFC9331)

Prague app

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

**Without L4S**
High RTT App (100ms)

Low RTT App (1ms)

Congestion

Loss

Rate Ratio = \( \frac{100\text{ms} + 1\text{ms}}{1\text{ms} + 1\text{ms}} = 50x \)

**With DCTCP ECN**
DCTCP app (100ms)

DCTCP app (1ms)

Immediate AQM (1ms)

Frequent marks to adapt rate and sustain low latency

Rate Ratio = \( \frac{100\text{ms} + 1\text{ms}}{1\text{ms} + 1\text{ms}} = 50x \)

RTT dependent:
- Rate \( \sim \frac{1}{\text{RTT}} \)
RTT unfairness: From NW ➔ Prague responsibility

**Without L4S**
- High RTT App (100ms)
- Rate Ratio = \( \frac{100ms + 20ms}{1ms + 20ms} \) ~ 6x
- Low RTT App (1ms)
- Network AQM (20ms)
- Congestion
- Loss

**With L4S**
- Prague app (100ms)
- Rate Ratio = \( \frac{100ms + 0ms}{1ms + 0ms} \) ~ 4x
- Prague app (1ms)
- Frequent marks to adapt rate and sustain low latency

**RTT dependent:**
- Bigger Q = more fairness

**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**
- X-Prague app (25ms)

**With Prague L4S**
- Prague app (25ms)

---

**Lowest Inertia gets bullied:**
- Large highly frequent variations

**Throughput reductions <25ms**

**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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>10</td>
<td>2.4</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>10</td>
<td>240</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>24</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1</td>
<td>240</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>1</td>
<td>2400</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.1</td>
<td>240</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0.1</td>
<td>2400</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>0.1</td>
<td>24000</td>
<td>10</td>
</tr>
</tbody>
</table>

High Rates possible
Not controllable under this rate with marks

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]

Data rate $r$

$$r \approx \frac{1}{p} - 1 \text{ [Mbps]}$$

Packet marking probability $p$

1 Gbps

100 Mbps

10 Mbps

1 Mbps

100 Kbps

100% 10% 1% 0.1% 0.01%

2 packets every 25 ms

2 mark every 25 ms

9 Mbps @ 10% marks

1 Mbps @ 50% marks

100 Mbps @ 1% marks

Packet marking probability $p$

1 Gbps

100 Mbps

10 Mbps

1 Mbps

100 Kbps

100% 10% 1% 0.1% 0.01%

2 packets every 25 ms

2 mark every 25 ms

9 Mbps @ 10% marks

1 Mbps @ 50% marks

100 Mbps @ 1% marks

Data rate $r$

$$r \approx \frac{1}{p} - 1 \text{ [Mbps]}$$

Packet marking probability $p$
Still compromises: Responsiveness, Rate, Latency, Smoothness

L4S flows

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

Classic flows

- 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

**Classic**
NW keeps buffers for Highest Possible Throughput

- BBR App
- Cubic App

**L4S**
NW keeps buffers empty for Lowest possible Latency

- Prague app

NW can optimize for each traffic type

- Prague app

Frequent marks to adapt rate and sustain low latency
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

- **Sender app**
  - Frame source
  - Encoder
  - Frame Packetizer & Pacer
  - Audio source
  - Audio Encoder
  - Storing only most recent frame

- **Frame size → Encoding rate**
- **Frames in Flight**
- **Pacing rate**

- **UDP-Prague per-Frame Prague CC**
  - Application protocol over UDP (standard or proprietary)
  - RTT, ACKs and CE-marking feedback in application layer

- **IP.ECN marking in network**

- **Receiver app**
  - Decode & Display
  - Client Receiver

- **Very low rates can be mixed in without latency**

- **L4S Prague update**
  - IP.ECN marking in network
  - ECN marking
Real-Time app optimization with UDP-Prague

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

UDP-Prague PragueCC uses ECN feedback directly to determine the pacing rate and window. The application protocol over UDP (standard or proprietary) interacts with the UDP stack and the IP ECN marking in the network. RTT, ACKs, and CE-marking feedback are used by the receiver app to decode and display the data.

Sender app:
- Data source
- Delivery budget per data source
- Scheduler
- Packetizer & Pacer

Receiver app:
- Packetizer & Pacer
- Client Receiver
- Decode & Display

Real-time optimization is achieved through the scheduling of data sources, packetization, and pacing, all integrated with the UDP stack and network feedback mechanisms.
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 minuscato/l4steam6.1.y
    - 6.6 minuscato/l4steam-6.6.y
    - 6.7 minuscato/net-next/three/upstream_l4steam
  - Rpi 6.6 minuscato/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** Iperf2 / 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