L4S and Prague update

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

Koen De Schepper
(koen.de_schepper@nokia-bell-labs.com)

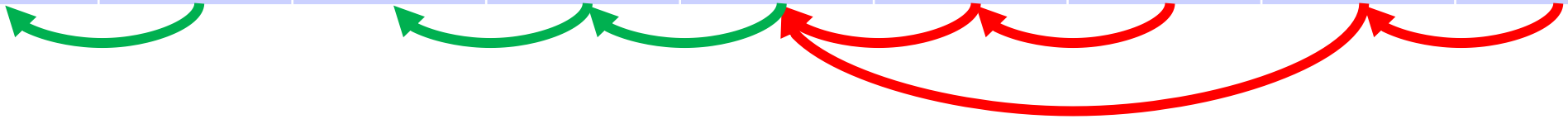
Chia-Yu Chang
(chia-yu.chang@nokia-bell-labs.com)

Contents

Linux Mainlining L4S

Started August 2024

Date	DualPI2 sched	DualPI2 tc iproute2	ACC_ECN 1/3	ACC_ECN 2/3	ACC_ECN 3/3	VirtIO spec	VirtIO driver	TCP-Prague tcp	TCP-Prague ss iproute2
Internal review	44 kernel patches 2 iproute2 patches 6 iterations								
Starting assumption	1 patch	1 patch	40 kernel patches			-	-	3 patches	1 patch
Patches	6	3	14 → 11 preparation	14 protocol	14 special cases	2	2	4	1
Iterations	v26	v12	v7	v16	v5	v10	v0	v0	v0
Net-next/ Linux-next	**July 2025**	**July 2025**	**March 2025**	**Sept 2025**					
mainline	**v6.17**	**v6.17**	**v6.15** **Not usable**	**(v6.18?)** **Not usable**	Activation of ACC_ECN	Updates in comments	Optimizes hypervisors	TCP-Prague	Stats for TCP-Prague
Status	**Success**	**Success**	**Success** 3 patches moved to 2/3	**Success** 1 patch moved to TCP-Prague	Ongoing: GRO packetdrill tests created	Blocked by ACC_ECN 3	Waiting for VirtIO Spec	Waiting for ACC_ECN	Waiting for TCP-Prague tcp



Accurate ECN field experience

Murphy…

Murphy was right

If it can happen it will happen

Be aware for too many “sanity” checks…

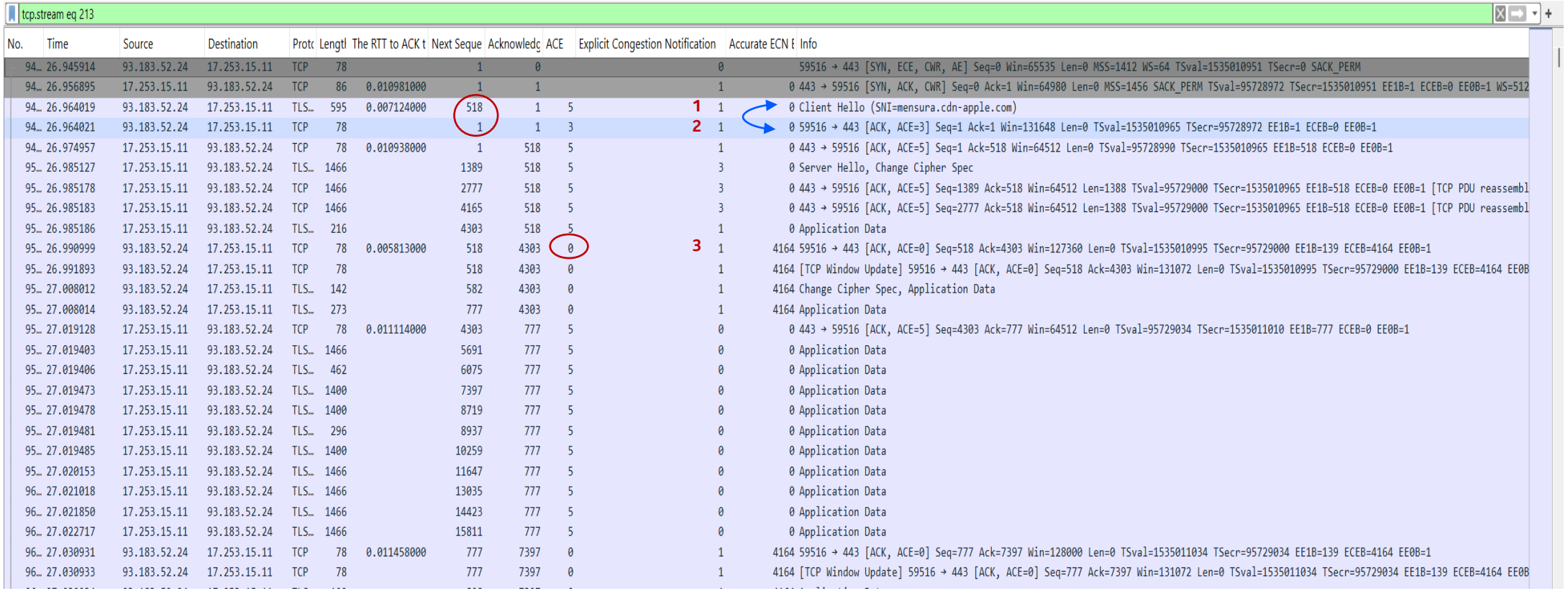
Especially if you can expect false positives

In real life tests they happened all the time…

Besides Murphy, also Stuart was right: tcpm

> I have reservations about this test. The idea that this heuristic might suffer from false positives and falsely disable Accurate ECN for no good reason worries me. A user is expecting the low-latency benefits of L4S and Accurate ECN, and then though no fault of their own sometimes does not get these benefits. I also have mixed feelings about the trade-off that is implicit here: If there are defective middleboxes on the path, would we like to see these middleboxes fixed, or do we want to enable them to remain broken in perpetuity (to the detriment of all the people unfortunate enough to have their traffic traverse those defective middleboxes)?

ACC_ECN false positive case more likely than broken implementations



tcp.stream eq 213

No.	Time	Source	Destination	Prot	Lengt	The RTT to ACK t	Next Seque	Acknowledg	ACE	Explicit Congestion Notification	Accurate ECN F	Info
94…	26.945914	93.183.52.24	17.253.15.11	TCP	78		1	0		0		59516 → 443 [SYN, ECE, CWR, AE] Seq=0 Win=65535 Len=0 MSS=1412 WS=64 TSval=1535010951 TSecr=0 SACK_PERM
94…	26.956895	17.253.15.11	93.183.52.24	TCP	86	0.010981000	1	1		1		0 443 → 59516 [SYN, ACK, CWR] Seq=0 Ack=1 Win=64980 Len=0 MSS=1456 SACK_PERM TSval=95728972 TSecr=1535010951 EE1B=1 ECEB=0 EE0B=1 WS=512
94…	26.964019	93.183.52.24	17.253.15.11	TLS…	595	0.007124000	518	1	5	1	1	0 Client Hello (SNI=mensura.cdn-apple.com)
94…	26.964021	93.183.52.24	17.253.15.11	TCP	78		1	1	3	2	1	0 59516 → 443 [ACK, ACE=3] Seq=1 Ack=1 Win=131648 Len=0 TSval=1535010965 TSecr=95728972 EE1B=1 ECEB=0 EE0B=1
94…	26.974957	17.253.15.11	93.183.52.24	TCP	78	0.010938000	1	518	5	1		0 443 → 59516 [ACK, ACE=5] Seq=1 Ack=518 Win=64512 Len=0 TSval=95728990 TSecr=1535010965 EE1B=518 ECEB=0 EE0B=1
95…	26.985127	17.253.15.11	93.183.52.24	TLS…	1466		1389	518	5	3		0 Server Hello, Change Cipher Spec
95…	26.985178	17.253.15.11	93.183.52.24	TCP	1466		2777	518	5	3		0 443 → 59516 [ACK, ACE=5] Seq=1389 Ack=518 Win=64512 Len=1388 TSval=95729000 TSecr=1535010965 EE1B=518 ECEB=0 EE0B=1 [TCP PDU reassembl
95…	26.985183	17.253.15.11	93.183.52.24	TCP	1466		4165	518	5	3		0 443 → 59516 [ACK, ACE=5] Seq=2777 Ack=518 Win=64512 Len=1388 TSval=95729000 TSecr=1535010965 EE1B=518 ECEB=0 EE0B=1 [TCP PDU reassembl
95…	26.985186	17.253.15.11	93.183.52.24	TLS…	216		4303	518	5	1		0 Application Data
95…	26.990999	93.183.52.24	17.253.15.11	TCP	78	0.005813000	518	4303	0	3	1	4164 59516 → 443 [ACK, ACE=0] Seq=518 Ack=4303 Win=127360 Len=0 TSval=1535010995 TSecr=95729000 EE1B=139 ECEB=4164 EE0B=1
95…	26.991893	93.183.52.24	17.253.15.11	TCP	78		518	4303	0	1		4164 [TCP Window Update] 59516 → 443 [ACK, ACE=0] Seq=518 Ack=4303 Win=131072 Len=0 TSval=1535010995 TSecr=95729000 EE1B=139 ECEB=4164 EE0B
95…	27.008012	93.183.52.24	17.253.15.11	TLS…	142		582	4303	0	1		4164 Change Cipher Spec, Application Data
95…	27.008014	93.183.52.24	17.253.15.11	TLS…	273		777	4303	0	1		4164 Application Data
95…	27.019128	17.253.15.11	93.183.52.24	TCP	78	0.011114000	4303	777	5	0		0 443 → 59516 [ACK, ACE=5] Seq=4303 Ack=777 Win=64512 Len=0 TSval=95729034 TSecr=1535011010 EE1B=777 ECEB=0 EE0B=1
95…	27.019403	17.253.15.11	93.183.52.24	TLS…	1466		5691	777	5	0		0 Application Data
95…	27.019406	17.253.15.11	93.183.52.24	TLS…	462		6075	777	5	0		0 Application Data
95…	27.019473	17.253.15.11	93.183.52.24	TLS…	1400		7397	777	5	0		0 Application Data
95…	27.019478	17.253.15.11	93.183.52.24	TLS…	1400		8719	777	5	0		0 Application Data
95…	27.019481	17.253.15.11	93.183.52.24	TLS…	296		8937	777	5	0		0 Application Data
95…	27.019485	17.253.15.11	93.183.52.24	TLS…	1400		10259	777	5	0		0 Application Data
95…	27.020153	17.253.15.11	93.183.52.24	TLS…	1466		11647	777	5	0		0 Application Data
96…	27.021018	17.253.15.11	93.183.52.24	TLS…	1466		13035	777	5	0		0 Application Data
96…	27.021850	17.253.15.11	93.183.52.24	TLS…	1466		14423	777	5	0		0 Application Data
96…	27.022717	17.253.15.11	93.183.52.24	TLS…	1466		15811	777	5	0		0 Application Data
96…	27.030931	93.183.52.24	17.253.15.11	TCP	78	0.011458000	777	7397	0	1		4164 59516 → 443 [ACK, ACE=0] Seq=777 Ack=7397 Win=128000 Len=0 TSval=1535011034 TSecr=95729034 EE1B=139 ECEB=4164 EE0B=1
96…	27.030933	93.183.52.24	17.253.15.11	TCP	78		777	7397	0	1		4164 [TCP Window Update] 59516 → 443 [ACK, ACE=0] Seq=777 Ack=7397 Win=131072 Len=0 TSval=1535011034 TSecr=95729034 EE1B=139 ECEB=4164 EE0B

Propose to remove this check in Linux (and others that can have false positives)

Do other implementations (Apple, FreeBSD, …) include such checks?

ACC_ECN RFC has a SHOULD and mentions the potential problem but now clear disadvantage is proven → errata on RFC?

Accurate ECN field experience

Impact of fallback

Measurement tools like RPM and iperf2/3 start TCP connections but are unaware of the fallbacks

Measurement results are a random mix of latency of L4S and Classic flows, not representative for the NW capabilities

Would be useful for Apps to get notified or to be able to query the L4S status of a socket

Extend socket API with a fallback reason (NW or receiver caused)?

Network improvements

WiFi: from EDCA to more deterministic MAC

DualPI2 for WiFi implementation guidelines published in WBA

Implementation Guidelines for Low Latency, Low Loss, and Scalable Throughput in Wi-Fi Networks (L4S) - Wireless Broadband Alliance

L4S is useful when the channel is fully occupied, so queuing latency occurs and can be controlled

WiFi MAC latency degrades with the number of devices actively sending data

→ more collisions and random exponential backoff → large latencies

WiFi WMM Video and Voice categories are only making things worse (designed for max one or 2 non-greedy users)

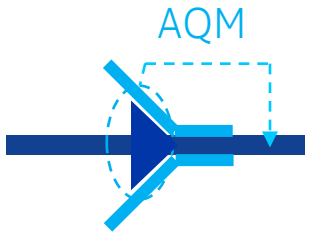
New EDCA parameters can be configured for BE that provide lower latencies for all under load

No need anymore for different categories with higher priorities (video and voice). The new EDCA works better.

Also, more deterministic MAC systems are possible (besides EDCA) but more awareness is needed in IEEE

Network improvements

Controlling rate without queue with ARM: Active Rate Management



Prague packets don't need to be delayed to rate control, so why build a queue and do AQM?

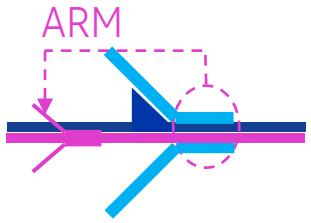
The 3 L's are most important for real-estate: Location, Location, Location, …
To achieve the 4 L's in L4S, the 3 H's are most important. It is all about: Headroom, Headroom, Headroom, …

Headroom gives L4S flows time to adapt, while schedulers block immediately and are intended to build queues.

Like DualPI2 with Classic traffic, the L4S traffic gets scheduling priority for low latency, but its rate is gently controlled by marks based on a per flow fair rate policy implemented by the coupled classic AQM.

Other policies are possible, including a static L4S rate budget (SRM: Static Rate Management) or per user fair aggregate rate policies (ARM).

For L4S, where possible Active Rate Management can be done without building the queue.



Where AQM or ARM is hard (on very high rate very aggregated networks), SRM is a very feasible alternative.

Rate Management has consequences

Certain assumptions are broken

Not a hard limit of capacity anymore: headroom is flexible wall that can be “pushed in”

- RPM tool increases the number of flows as long as it can get more throughput
- This can drive the network in overload before no more (significant) throughput is achieved
- Same problem happens if other CCed flows are active in the same bottleneck
- When should a measurement tool stop adding more flows to prevent overload situations?
- Test tools should be usable next to real applications, without driving the NW in overload

Unresponsive traffic will not necessarily get queued and delayed anymore

- No delay with Rate Management
- Loss might be applied as alternative to delay

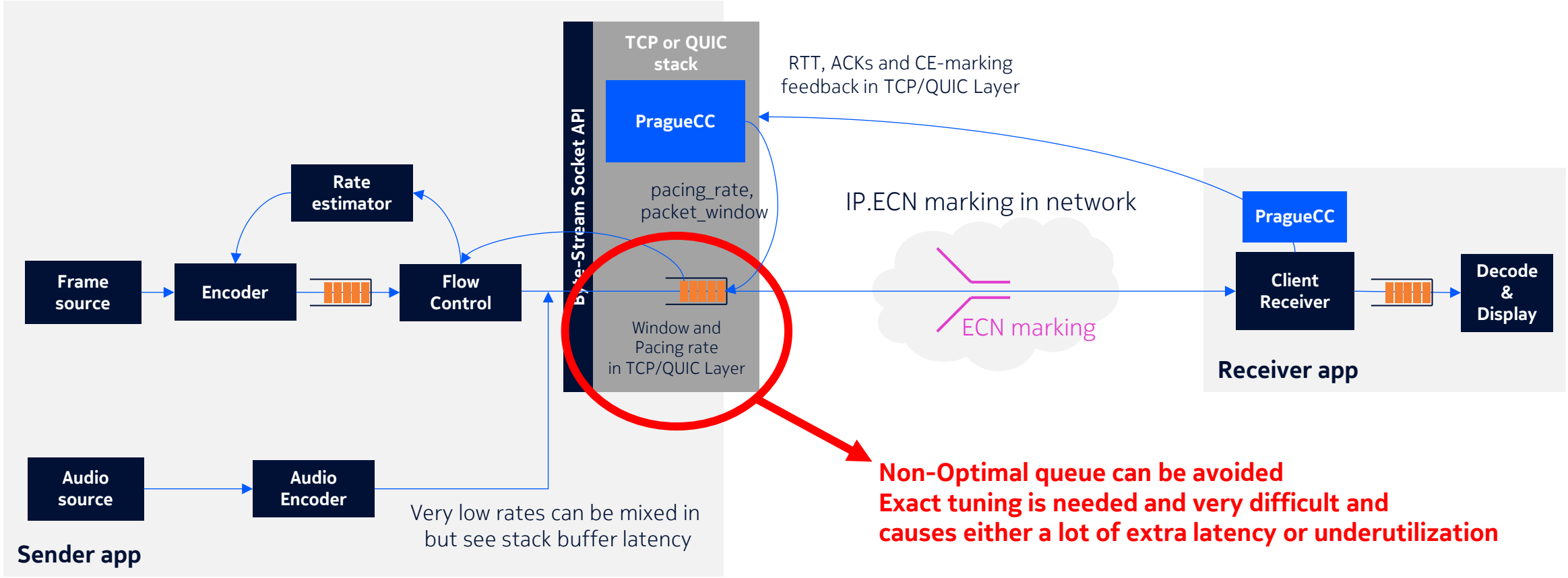
PragueCC for interactive apps

GitHub repository: L4STeam/udp_prague

- Prague congestion control protocol for UDP-based applications targeting very interactive user experience
- Single source C++ prague_cc.h/cpp files define the PragueCC object with minimal dependencies
- Platform independent code (Windows, Linux, Android, FreeBSD, Apple, …)
- Allows evolution of Prague without impacting the API (API can still evolve for now)
- Performs both Sender (Data) and Receiver (ACK) functions, supporting bidirectional data transfers
- Allows directly controlling application data generation rate and non-blocking delivery
- Example load generators present for sender and receiver (either run in client or server mode)
- Iperf2 integration at: Iperf 2 / Code / udp_prague, and ns-3 coming soon (currently in a private git)

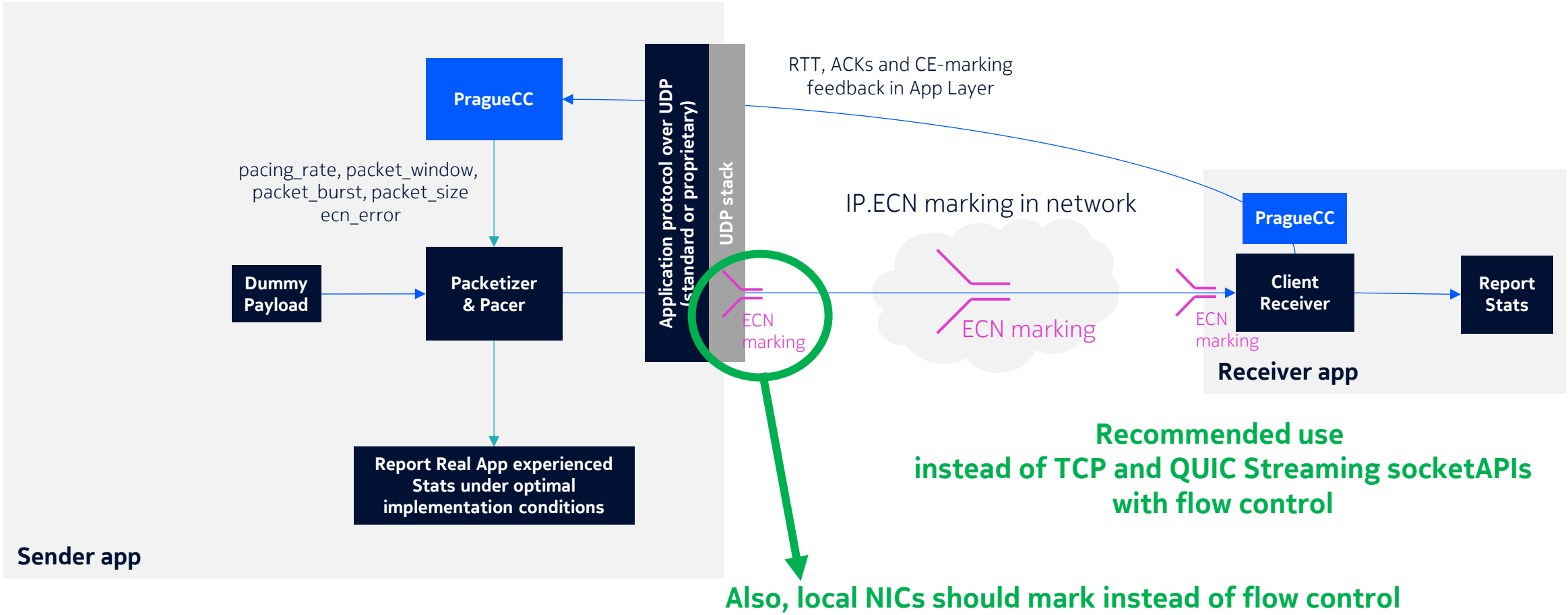
PragueCC with TCP or QUIC stacks

Stacks can use PragueCC object, but need non-optimal flow control



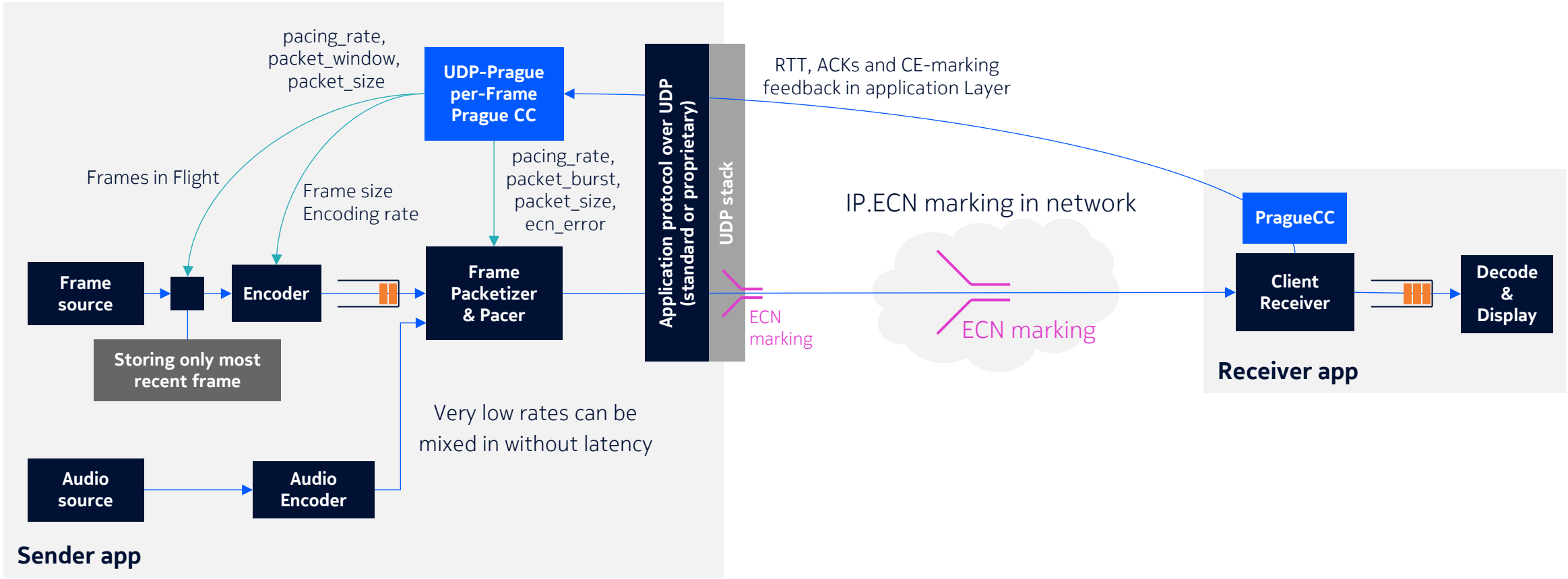
No-Queue use of PragueCC in UDP-Prague, iperf2 and ns3

Load generator tools can use PragueCC without other latencies and can report exact Stats



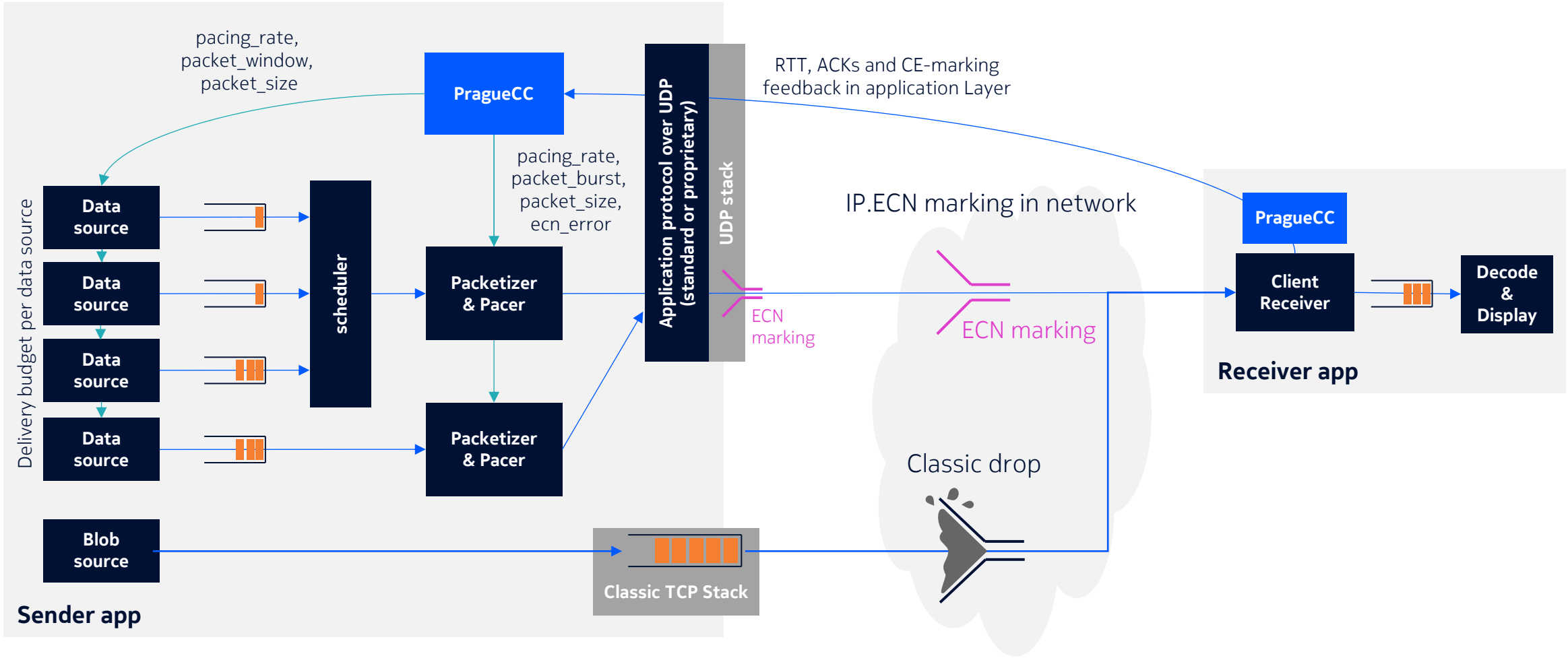
Real-Time app optimization with PragueCC

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



Real-Time app optimization with PragueCC

PragueCC uses ECN feedback directly to control the data source(s)



TSVWG work?

ACC_ECN errata?

Should we risk false positives at the “cost” of losing the L4S service

Should we apps fall back at all ? Should we “correct” (or cover for) network errors, or should we let it to the NW to fix them.

What info to provide to apps? Which APIs?

Do we want to document alternative NW mechanisms (like Rate Management) in TSVWG?

Koen De Schepper
(koen.de_schepper@nokia-bell-labs.com)

Chia-Yu Chang
(chia-yu.chang@nokia-bell-labs.com)