UiO : **Faculty of Mathematics and Natural Sciences**
University of Oslo

End-to-end is a dead end

(for Internet transport !)

Michael Welzl
MIR3, 12. 9. 2022

A quick disclaimer

- I do *not* claim that the end-to-end argument is wrong
  - But it's been interpreted too strictly
  - It says: "think twice" before putting complexity in the network
  - Especially, let applications do application-specific functions

- Routing can be arbitrarily complex, *in the network*
  - (no I'm not going to discuss SCION with you)

- …but congestion also happens in the network!
  - Doing it only end-to-end was ok as a quick fix, but it's like doing *only* source routing, always and forever

Another disclaimer ☺

- These are my two “outrageous opinions”
  - They go together into one proposal
- I believe this has a lot of potential
  - And I would love it if we could discuss it further
  - I’m very thankful for this long slot!
- I also hope for help (hey, I have come a long way!) ….
  - I dream of ERC, but is this <u>risky</u> enough?
    - Crazy, unusual = high risk?

Disclaimer: no more disclaimers

Which problem am I trying to solve?

- A wireless link's capacity can fluctuate fast & drastically…
  - Side note: fluctuating traffic can have a similar effect
    … but let's stick with wireless (a real problem & easier to think about)

- … and end-to-end congestion controls don't cope well.
  - E.g., paper #1 shows this for [Cubic, L4S (TCP Prague), BBRv1, BBRv2] over mmWave
  - E.g., paper #2 shows that TCP with a PEP is faster than QUIC over a satellite link

I call this a "dilemma".
It's called "QUIC", after all.

Paper #1: *Srivastava, F. Fund and S. S. Panwar, "An Experimental Evaluation of Low Latency Congestion Control for mmWave Links," IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2020*

Paper #2: *Kuhn, N, Michel, F, Thomas, L, et al. QUIC: Opportunities and threats in SATCOM. Int J Satell Commun Network. 2021; 1- 13. doi:10.1002/sat.1432*

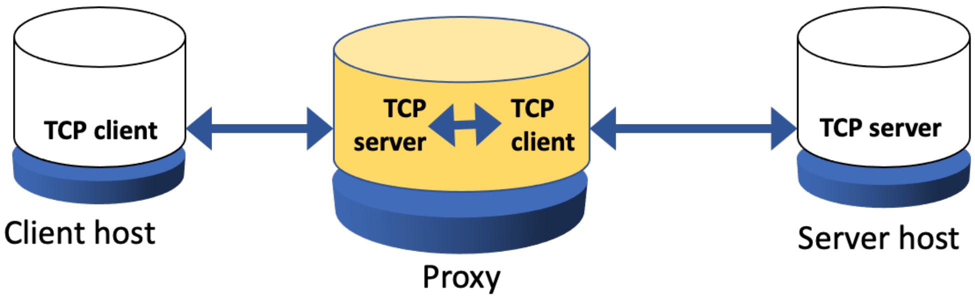
Three reasons

1. (End-to-end) sender gets the signal late => reacts late

2. (End-to-end) sender lacks key information
   - E.g. "*More capacity is now available.*"
   - Nokia's "throughput guidance" informs e2e control loop about capacity growth, but such ideas never succeed in the IETF
   - Several failure reasons; most importantly: **"how can the sender be sure that the signal comes from the bottleneck?"**
     - *Side note: "throughput guidance" has then been included in ETSI MEC, where this may make more sense*

3. When more capacity becomes available, there may not yet be enough data near the bottleneck
   - So we may miss the "transmission opportunity"

TCP Connection-splitting PEPs <u>almost</u> solve these problems



1. (End-to-end) sender gets the signal late => reacts late
   – **Fixed:** PEP executes local control

2. Sender lacks key information
   – **Fixed:** PEP is where the problem happens, can obtain information and even implement customized control (done for satellite comm.)

3. When more capacity becomes available, there may not yet be enough data near the bottleneck
   – **<u>Almost</u> fixed:** PEP can, in principle, ask for data at any time…

…. yet, such PEPs are bad!

1. **Terrible:**
   - Ossification
     - Hard to upgrade TCP because of the assumptions that PEPs make
   - Unknown behavior: good or bad? For which application?
     - PEPs are not part of the design, hence they must “cheat” TCP, with possibly unexpected / undesirable results

2. **Not good enough:** getting data to the bottleneck
   - TCP ACKs convey two things:
     1. ***"Give me more data"*** (**good**)
     2. ***"I received a packet, delete the data from your send buffer"*** (**bad**)
   - PEP can only state both, needs to limit how much it asks for
   - If a PEP buffers a lot, it produces a lot of delay, which matters for some applications … but the PEP cannot differentiate (or very limited: e.g. port, ..)

I believe that we can get this right.

Back to my “outrageous opinions”…

1) "We should use large queues in the network to reduce latency!"

- Imagine a PEP-like device at the bottleneck asking for a *lot* of data, early (long before the receiver gets it), and:

    1. Not claiming reliability (sender keeps data in its buffer until an ACK arrives from the real receiver)
        - This enables the PEP to throw packets away, like a regular router
    2. Only doing this for applications that benefit from it
        - With 1), this enables the PEP to ask for an unlimited amount of data

- **What are these applications?**

Which apps may want large buffers?

- Everything that is “fire and forget” !
  - One-way streaming, web surfing, …
  - Note: latency is not only per-packet sojourn time
    - Buffers can help whenever the FCT matters
    - E.g., web surfing = latency critical, yet it can benefit from this!

- **NOT:** applications that need to stay in control of their data until the last moment (interactive applications)
  - Maybe we don’t need to rule them out forever...
    ...but for now, we do.

How could this be done?

- PEP tells the sender: "***give me more, but don't yet throw away***"
  - This must be standardized, and the PEP must be known and authenticated

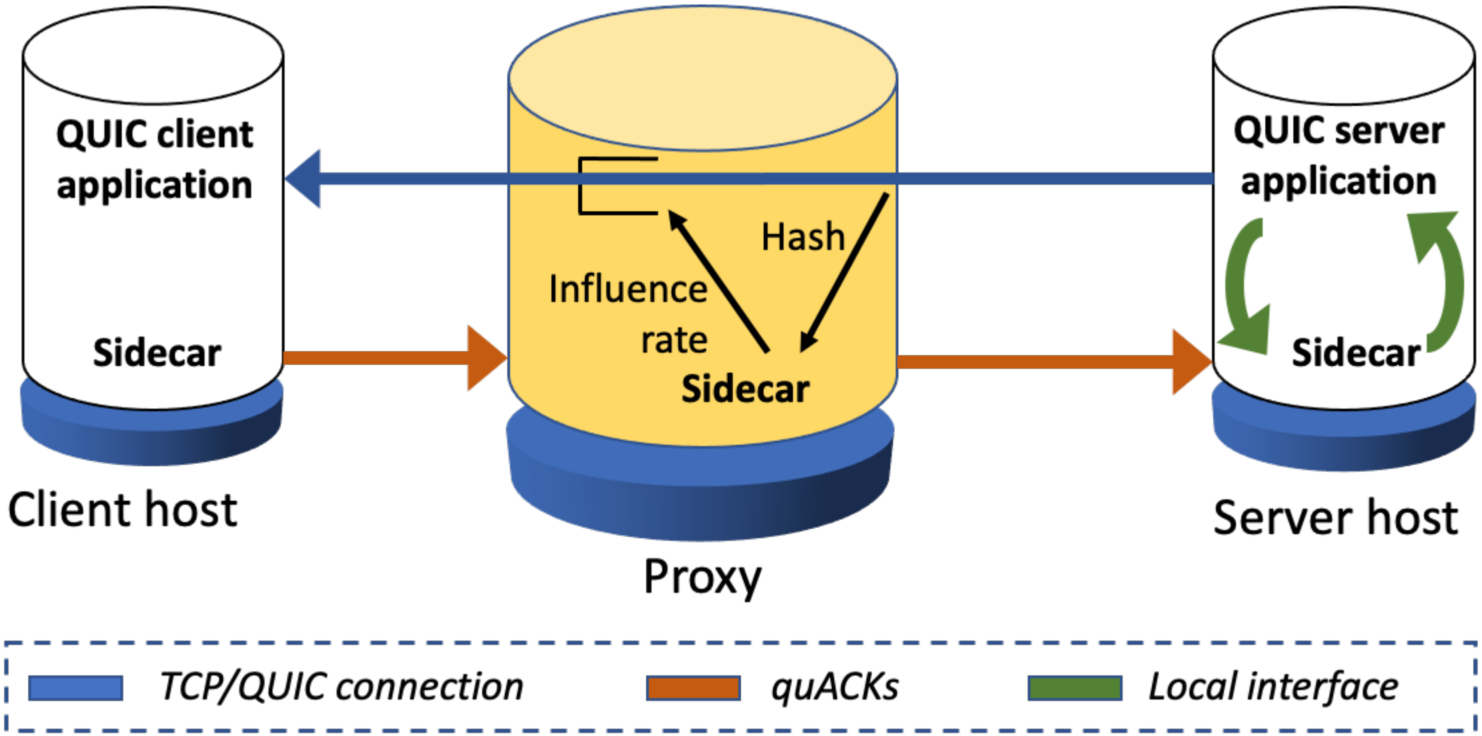
- **This brings me to my outrageous opinion #2:**
  "Proxies (PEPs) are the way to go, also for QUIC"

- Next: "Sidecar"
  Suggested non-transparent PEP design for (not only!) QUIC
  - Collaboration with Keith Winstein
  - Presented at *PANRG, IETF-113, Vienna, March 2022*
  - *G. Yuan, D. K. Zhang, M. Sotoudeh, M. Welzl, K. Winstein: "Sidecar: In-Network Performance Enhancements in the Age of Paranoid Transport Protocols", accepted for publication, ACM HotNets 2022.*

Separation of concerns

- A separate “sidecar (SC)” for non-critical PEP functions, independent of main protocol
  - Main protocol chooses services ("opt-in") over a local sidecar interface (on the same host)
  - SC can help QUIC, TCP... as main protocol
- Minimize changes to main protocol
  - No connection splitting, SC proxy does not parse the header
- Sidecar ossification then means: the PEP function does not improve further (*disappointing but harmless*)
- PEP functions are SC use cases ("SC protocols")

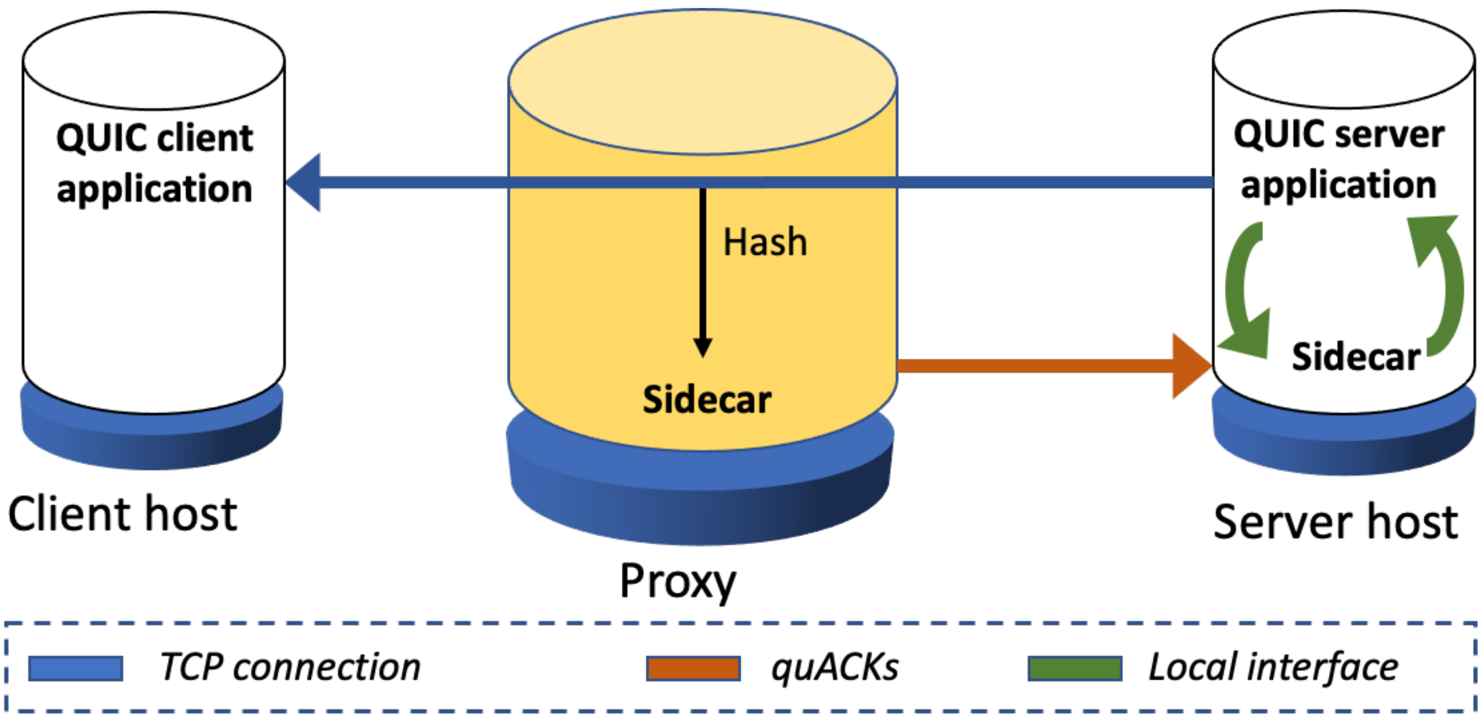
CC division



- Get data to the proxy early; drain data from the proxy buffer using the proxy-client CC loop
- Only needs a server-side change

ACK reduction for wireless links

Dzmitry Kliazovich, Simone Redana, Fabrizio Granelli: "Cross-layer error recovery in wireless access networks: The ARQ proxy approach", International Journal of Communication Systems, Vol. 25, n. 4, pp 213-222, 2011



- Server tells client via the main protocol that it should send fewer positive ACKs

QuACKs

- Could be piggybacked (QUIC: UDP options)
- Could use a hash over transport header bytes
  - Difficult to define efficient cumulative ACKs
- Could be done with power sums
  - Sending the first *k* power sums in a QuACK informs the sender about up to *k* missing elements
  - Analyzed for efficiency in our paper

Conclusion

- **We believe that this is a viable way forward to solve the e2e encryption / ossification / PEP dilemma.**

- Research needed to make it work
  - Different from transparent PEPs, SC entities must find each other
  - SC proxy can just send QuACKs; doesn't need to trust anyone
  - Sender-side SC entity needs to trust the SC proxy... but QuACKs are perhaps not easy to produce for an adversary
  - Path changes: if there's a different SC proxy on the new path, it just begins to send QuACKs
  - but need a setup phase, or we could get N QuACKing SC proxies on a path

- Research opportunities
  - We now have more data available in the network!
  - Use for ... e.g., better multipath.

Thank you!

Questions, comments?