Service Awareness rather than Path Awareness

Michael Welzl (University of Oslo), Keith Winstein (Stanford University)
The problem

• Performance Enhancing Proxies (PEPs) sometimes let TCP work better than QUIC, e.g. over satellites
  – Has anyone tried QUIC over mmWave?
  – Encrypting transport headers solves ossification, but at a cost

• PEPs are not strictly evil
  – They try to be useful, and sometimes succeed
  – Claim: ossification is at least partially due to transparent proxy design
    (they must cheat, so they must make assumptions about header fields)

• MASQUE is not transparent. Add PEP functions there?
  – Maybe? But this might cause ossification problems again...
  – Perhaps depending on the PEP function
What we suggest

• Separation of concerns
  – A separate “sidecar (SC)” protocol for non-critical PEP functions, independent of main protocol
  – Non-criticality ensured by letting main protocol choose services over a local sidecar interface (on the same host!)

• Minimize changes to “main” protocol
  – Sidecar ossification means: the PEP function does not improve further (bad but harmless)

• PEP functions are use cases of the sidecar protocol
Sidecar functionality

• **Data plane**: directly affect main protocol
  – *Without parsing header*: queue management, re-transmission, ..

• **Control plane**
  – Local (on host) information exchange with main protocol

• **SC ACKs between sidecar entities**
  – Hash over main protocol’s transport header
  – SC ACKs are either separate or piggybacked (e.g., QUIC: UDP options)

• **Next: two example use cases**
  – Written “QUIC”, but should work the same way for, e.g., TCP or SCTP
Example use case 1: link-specific congestion control

- Adjacent to fluctuating-capacity link
  - SC proxy’s congestion control should track available capacity better, and needs data packets for when capacity becomes available.
Example use case 1: link-specific congestion control

- Adjacent to fluctuating-capacity link
  - SC proxy’s congestion control should track available capacity better, and needs data packets for when capacity becomes available

Service choice: If you tell me to increase my cwnd, I will.
Example use case 1: link-specific congestion control

- Adjacent to fluctuating-capacity link
  - SC proxy’s congestion control should track available capacity better, and needs data packets for when capacity becomes available

*Service choice:* If you tell me to increase my cwnd, I will.

*SC notification:* An ACK has arrived, increase your cwnd.
Example use case 1:
link-specific congestion control

- Adjacent to fluctuating-capacity link
  - SC proxy’s congestion control should track available capacity better, and needs data packets for when capacity becomes available

---

**Service choice:**
If you tell me to increase my cwnd, I will.

**SC notification:**
An ACK has arrived, increase your cwnd.

**Minimal changes to QUIC: only server side**
- Local interface communication
- If the service is chosen: increase cwnd when SC tells us to
Example use case 2: WiFi AP ACKs on behalf of host

- AP keeps track of transport packet – ACK mapping
- Creates a transport ACK to the server when LL ACK from client arrives
  - Client can send fewer transport ACKs. Reduces overhead: fewer collisions, less power wasted

Example use case 2:
WiFi AP ACKs on behalf of host

- AP keeps track of transport packet – ACK mapping
- Creates a transport ACK to the server when LL ACK from client arrives
  - Client can send fewer transport ACKs. Reduces overhead: fewer collisions, less power wasted

Example use case 2: WiFi AP ACKs on behalf of host

- AP keeps track of transport packet – ACK mapping
- Creates a transport ACK to the server when LL ACK from client arrives
  - Client can send fewer transport ACKs. Reduces overhead: fewer collisions, less power wasted

Example use case 2:
WiFi AP ACKs on behalf of host

- AP keeps track of transport packet – ACK mapping
- Creates a transport ACK to the server when LL ACK from client arrives
  - Client can send fewer transport ACKs. Reduces overhead: fewer collisions, less power wasted


---

**Service choice:** I will treat SC ACKs like client ACKs (but keep data in the send buffer until client cumulatively ACKs, just in case).

**SC notification:** An ACK has arrived.

**Minimal changes to QUIC: server + client side**
- Server: local interface communication
- If the service is chosen:
  - Via QUIC, tell client to send fewer ACKs
  - Accept ACKs from SC, but keep data in send buffer
Conclusion

• We believe this is a way forward to solve the e2e encryption / ossification / PEP dilemma

• Research needed
  – How to limit hashing / SC ACK overhead?
  – The devil is in the details: what are viable use cases?
    • E.g., link-specific congestion control use case: different from TCP connection splitter, SC entities must find and trust each other
      – SC proxy can just send SC ACKs back towards the sender; doesn’t need to trust anyone
      – Sender-side SC entity needs to trust the SC proxy... but the SC proxy can’t easily guess hashes
      – Path changes: if there’s a different SC proxy on the new path, it just begins to send SC ACKs
      – ... but there needs to be a setup phase, or else we could get N SC proxies on a path, all ACKing 😊
  – Can this really be done independent of the main protocol?
Thanks!

Questions?