Sidekick: In-Network Assistance for Secure End-to-End Transport Protocols

Gina Yuan, Matthew Sotoudeh, David K. Zhang, Michael Welzl\textsuperscript{+}, David Mazières, Keith Winstein

NSDI ’24 (April 16-18, 2024)
Stanford University & \textsuperscript{+}University of Oslo
Imagine using the Wi-Fi on a train.
The Wi-Fi seems really bad.

Except for TCP?
TCP is faster than QUIC!?
QUIC (and WebRTC) are *encrypted on the wire.*

<table>
<thead>
<tr>
<th>QUIC Segment</th>
<th>vs.</th>
<th>TCP Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>!@$%^ (Encrypted Data)</td>
<td>TCP Header (visible to middleboxes)</td>
<td>Source Port</td>
</tr>
<tr>
<td>“secure” transport protocol</td>
<td>Acknowledgment number</td>
<td>Sequence number</td>
</tr>
<tr>
<td>TCP Payload (encrypted)</td>
<td>DO</td>
<td>Flags</td>
</tr>
<tr>
<td></td>
<td>RSV</td>
<td>Window</td>
</tr>
<tr>
<td></td>
<td>Checksum</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Options</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLS-Encrypted Data</td>
</tr>
</tbody>
</table>
TCP is *unencrypted* on the wire.

```
QUIC Segment

!@#$%^ (Encrypted Data)

"secure" transport protocol
```

```
TCP Segment

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment number</td>
<td></td>
</tr>
<tr>
<td>DO</td>
<td>RSV</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>Options</td>
<td></td>
</tr>
<tr>
<td>TLS-Encrypted Data</td>
<td></td>
</tr>
</tbody>
</table>

TCP Header (visible to middleboxes)

TCP Payload (encrypted)
Transport is *end-to-end* for secure protocols.

However, the router divides the network path into **two distinct path segments**.
In contrast, middleboxes can (and do) help TCP.

- Faster retransmissions and a better congestion response.
- 20-40% of Internet paths, most cellular paths, contain a TCP PEP [Honda et. al., 2011; Edeline & Donnett, 2019]
Performance-enhancing proxies have a dark side...

- Ossification of existing protocols
- Ossification of future protocols
Performance-enhancing proxies have a dark side...

- Ossification of existing protocols
- Ossification of future protocols
- **Today:** encrypt the transport layer (avoiding ossification), but give up on PEPs
Can there be a universal PEP for *Arbitrary* transport protocols?
Sidekick protocols: in-network assistance that leaves the base protocol unchanged *on the wire*.

1. Sidekick protocols on an *adjacent* connection
What *useful* information can a middlebox send for random-looking packets?

Easy problem for cleartext TCP sequence numbers...
What *useful* information can a middlebox send for random-looking packets?

**Identifier:**  \[ \text{hash( } \ ) \in [0, 2^{32}) \]

But for random-looking packets?
What *useful* information can a middlebox send for random-looking packets?

1. Sidekick protocols on an *adjacent* connection
2. QuACKs = concise, efficient ACKs of random packets

*not QUIC ack, since other protocols are ok too!*

quACK = quick* ACK
And what should the *sender* do to obtain a performance benefit for its base connection?

1. Sidekick protocols on an *adjacent* connection
2. QuACKs = concise, efficient ACKs of random packets
3. Path-aware sender behavior: retransmission, congestion control, flow control

Note: *Arbitrary* base protocol.

No reliability guarantee, unlike a TCP ACK. QuACKs describe *which* packets are received and *where*. 

16
The rest of the talk...

1. Sidekick protocols on an *adjacent* connection
2. QuACKs = concise, efficient ACKs of random packets
3. Path-aware sender behavior: retransmission, congestion control, flow control
   + implementation & eval
Desired quACK properties

(2b) Efficient to decode $R \subseteq S$, given $S$ and a quACK

(2a) Efficient to process a single packet

(1) Concise: low link overhead

(3) Loss-Resilient: cumulative representation

quACK = ACK of random packets
How can we construct a quACK with these properties?

<table>
<thead>
<tr>
<th>Description</th>
<th>Strawman 1</th>
<th>Strawman 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Echo every identifier.</td>
<td></td>
<td>Hash a sorted concatenation of every identifier.</td>
</tr>
<tr>
<td>SHA256(85</td>
<td>. . .</td>
<td>944)</td>
</tr>
<tr>
<td>Encode Time</td>
<td>0</td>
<td>27 ns/pkt</td>
</tr>
<tr>
<td>Decode Time</td>
<td>0</td>
<td>830 ms/quACK</td>
</tr>
<tr>
<td>QuACK Size</td>
<td>25 pkts × 4 bytes/pkt</td>
<td>36 bytes</td>
</tr>
<tr>
<td>Loss-Resilient</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Parameters: 25 outstanding packets, up to $t = 10$ missing packets, 32-bit identifiers
# Power sum solution

<table>
<thead>
<tr>
<th>Strawman 1</th>
<th>Strawman 2</th>
<th>Power Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Echo every identifier.</td>
<td>Hash a sorted concatenation of every identifier.</td>
</tr>
<tr>
<td><strong>Encode Time</strong></td>
<td>0</td>
<td>27 ns/pkt</td>
</tr>
<tr>
<td><strong>Decode Time</strong></td>
<td>0</td>
<td>830 ms/quACK</td>
</tr>
<tr>
<td><strong>QuACK Size</strong></td>
<td>25 pkts × 4 bytes/pkt</td>
<td>36 bytes</td>
</tr>
<tr>
<td><strong>Loss-Resilient</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Parameters: 25 outstanding packets, up to $t = 10$ missing packets, 32-bit identifiers

---

State maintained in the sidekick protocol

**Middlebox maintains**

a threshold number of power sums of the received packets \( R \subseteq S \)

- Threshold \( t \) = upper bound on the number of missing packets
- The last packet received
- The number of packets

\[
4^t + 8 \text{ bytes}
\]

**Sender maintains**

a threshold number of power sums of the sent packet identifiers \( S \)

- A log of the sent packets

\[
\sum_{x \in R} x^n \quad \text{(finite field)}
\]
Mathematical Intuition: Decoding QuACKs

**Sender Goal:** decode R or $S \setminus R$ given a quACK

one missing packet: \[ \sum_{x \in S} x - \sum_{x \in R} x = \sum_{x \in S \setminus R} x. \] \[ \text{sender state} \quad \text{receiver state} \]

$m$ missing packets: \[ \left\{ \sum_{x \in S \setminus R} x^i = d_i \mid i \in [1,m] \right\} \]

**Intuition:** Solve a system of $m$ polynomial equations in $m$ variables, where $m \leq$ a threshold $t$. The solutions are the missing packets.
Talk Outline

1. Sidekick protocols on an *adjacent* connection
2. QuACKs = concise, efficient ACKs of random packets
3. Path-aware sender behavior: retransmission, congestion control, flow control
   + implementation & eval
Background: congestion control

CUBIC modulates a *congestion window* (cwnd), using loss from ACKs as a signal.
From before: **QUIC** and **TCP** both used CUBIC.

- **QUIC**: one end-to-end CUBIC cwnd
- **TCP**: two connections, each with its own CUBIC cwnd

### Experiment Parameters

- **Loss % on Wireless Link (mininet emulation)**
- **Goodput (Mbit/s)**
- **1ms delay**
- **100 Mbit/s**
- **X% loss**
- **25ms delay**
- **10 Mbit/s**
- **0% loss**

![Graph showing Goodput vs Loss % on Wireless Link](image)

- **TCP (Split CUBIC)**
- **QUIC (E2E CUBIC)**
- **TCP (E2E CUBIC)**
What should be the congestion response to loss from *quACKs* to obtain a performance benefit?

![Graph showing goodput vs. loss percentage on a wireless link.](image)

- **Loss % on Wireless Link:** (mininet emulation)
- **Goodput (Mbit/s):**
  - TCP (Split CUBIC)
  - QUIC (E2E CUBIC)
  - TCP (E2E CUBIC)

**Experiment Parameters**

- 1ms delay
- 100 Mbit/s
- X% loss
- 25ms delay
- 10 Mbit/s
- 0% loss

**Option 1:** Same as end-to-end. *(Just as slow. 😞)*
What should be the congestion response to loss from quACKs to obtain a performance benefit?

Option 2: Nothing. (Not fair, will CAUSE congestion collapse. 😞)

Experiment Parameters

<table>
<thead>
<tr>
<th>Bottleneck Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1ms delay 100 Mbit/s</td>
</tr>
</tbody>
</table>

- **TCP (Split CUBIC)**
- **QUIC (E2E CUBIC)**
- **TCP (E2E CUBIC)**
What should be the congestion response to loss from *quACKs* to obtain a performance benefit?

**Experiment Parameters**

- Loss % on Wireless Link ([mininet emulation])
- Goodput (Mbit/s)
- Bottleneck Capacity

**Graph Details**

- TCP (Split CUBIC)
- QUIC (E2E CUBIC)
- TCP (E2E CUBIC)
- QUIC (E2E Path-Aware CUBIC)

**Graph Description**

- Loss % on Wireless Link: 0% to 8%
- Goodput (Mbit/s): 0 to 8

**Options for Congestion Response**

- Option 3: As fast/fair as a TCP PEP! 😊 (Or close, while being end-to-end.)
Mathematical Intuition: Path-aware CUBIC

**Idea:** Update the portion of the end-to-end cwnd that corresponds to the path segment of the last congestion event.

**Algorithm:** \[ \beta = 1 - r(1 - \beta^*) \] and \[ C = \frac{C^*}{r^3}. \]

- \( r \) = RTT of the path segment of the last congestion event / end-to-end RTT
- \( \beta \) = multiplicative decrease scaling factor in CUBIC
- \( C \) = cubic growth function scaling factor in CUBIC

**Intuition:** end-to-end PACUBIC cwnd ≈ the sum of the split CUBIC cwnds
Talk Outline

1. Sidekick protocols on an adjacent connection
2. QuACKs = concise, efficient ACKs of random packets
3. Path-aware sender behavior: retransmission, congestion control, flow control
   + implementation & eval
Implementation

<table>
<thead>
<tr>
<th>Module</th>
<th>Language</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuACK library</td>
<td>Rust</td>
<td>1772</td>
</tr>
<tr>
<td>Middlebox sidekick binary</td>
<td>Rust</td>
<td>833</td>
</tr>
<tr>
<td>quiche client integration</td>
<td>Rust</td>
<td>1821</td>
</tr>
<tr>
<td>libcurl client integration</td>
<td>C</td>
<td>1459</td>
</tr>
<tr>
<td>Media server/client + integration</td>
<td>Rust</td>
<td>478</td>
</tr>
</tbody>
</table>

https://github.com/ygina/sidekick

NSDI ‘24 Community Award!
Middlebox binary

- **74 cycles/pkt** (0.9%) to encode an identifier into a $t=10$ quACK
- Largest overhead was reading the packet contents from the network interface (97.5% of cycles/pkt)
- Max achieved throughput for a single core was **464k pkts/s** on a 2.30 GHz CPU
Client integrations

• Simple media client/server
  • ~150 additional LOC (Rust) to utilize sidekick protocols for retransmission

• QUIC+HTTP/3 production client/server
  • ~1500 additional LOC (C) to establish sidekick connection in libcurl client
  • ~1800 additional LOC (Rust) to implement retransmission, congestion control, and flow control logic in Cloudflare quiche
  • Overhead: 3% more packets, quACKs and ACKs have similar processing time
Applications (Emulation)

Scenario #1: Low-Latency Media (simple media protocol)
Goal: Reduce de-jitter buffer tail latency.

Scenario #2: Connection-Splitting PEP Emulation (HTTP/3+QUIC)
Goal: Achieve high throughput while being as fair as TCP PEPs.

Scenario #3: ACK Reduction (HTTP/3+QUIC)
Goal: Reduce the ACK frequency of a receiver to save energy.

Various Parameters (see paper)
Applications (Real World)

Scenario #1: Low-Latency Media
Reduced the 99th percentile de-jitter buffer latency by 91%.

Scenario #2: Connection-Splitting PEP Emulation
Improved the speed of a 50 MB HTTP/3 upload by 50%.
**Conclusion**

Sidekick protocols provide in-network assistance to arbitrary base protocols. QuACKs enable senders to emulate PEPs while leaving the protocol free to evolve.

[https://www.github.com/ygina/sidekick/](https://www.github.com/ygina/sidekick/)

Gina Yuan  Matthew Sotoudeh  David K. Zhang  Michael Welzl  David Mazières  Keith Winstein