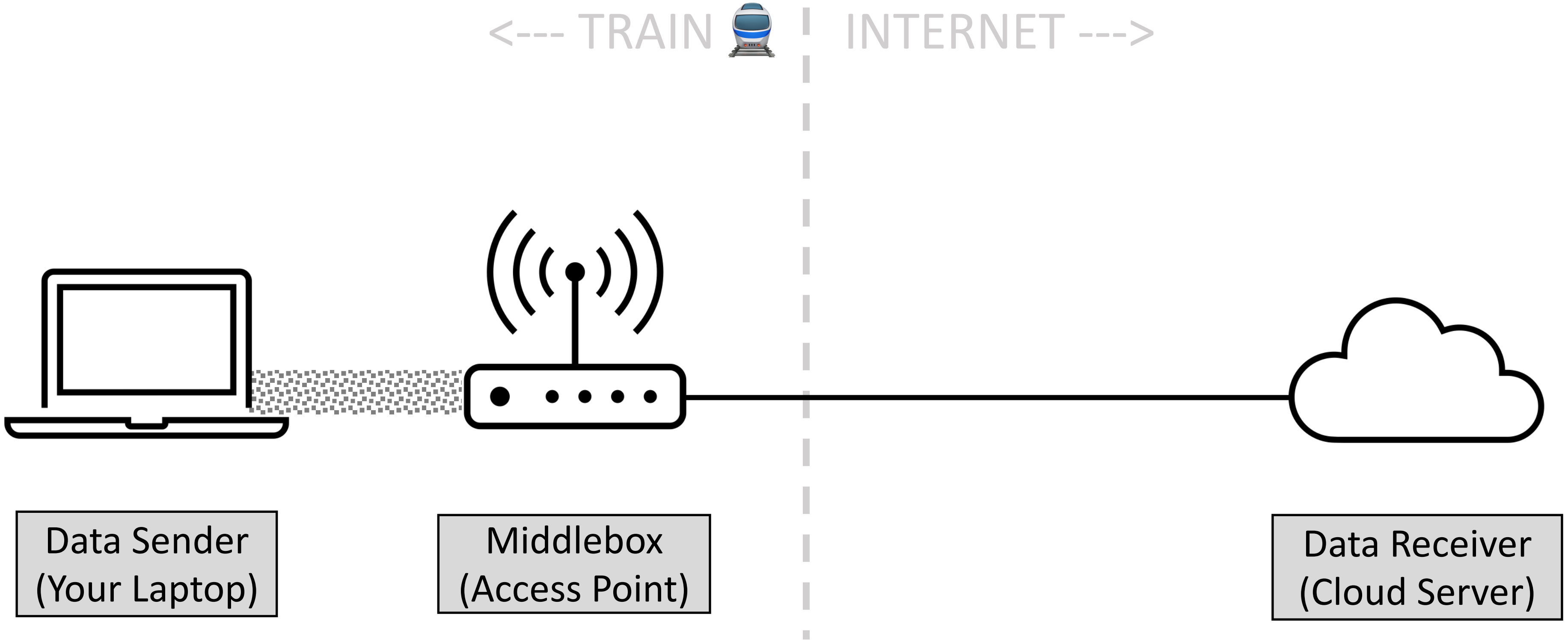
Sidekick: In-Network Assistance for Secure End-to-End Transport Protocols

Gina Yuan, Matthew Sotoudeh, David K. Zhang,
Michael Welzl[+], David Mazières, Keith Winstein

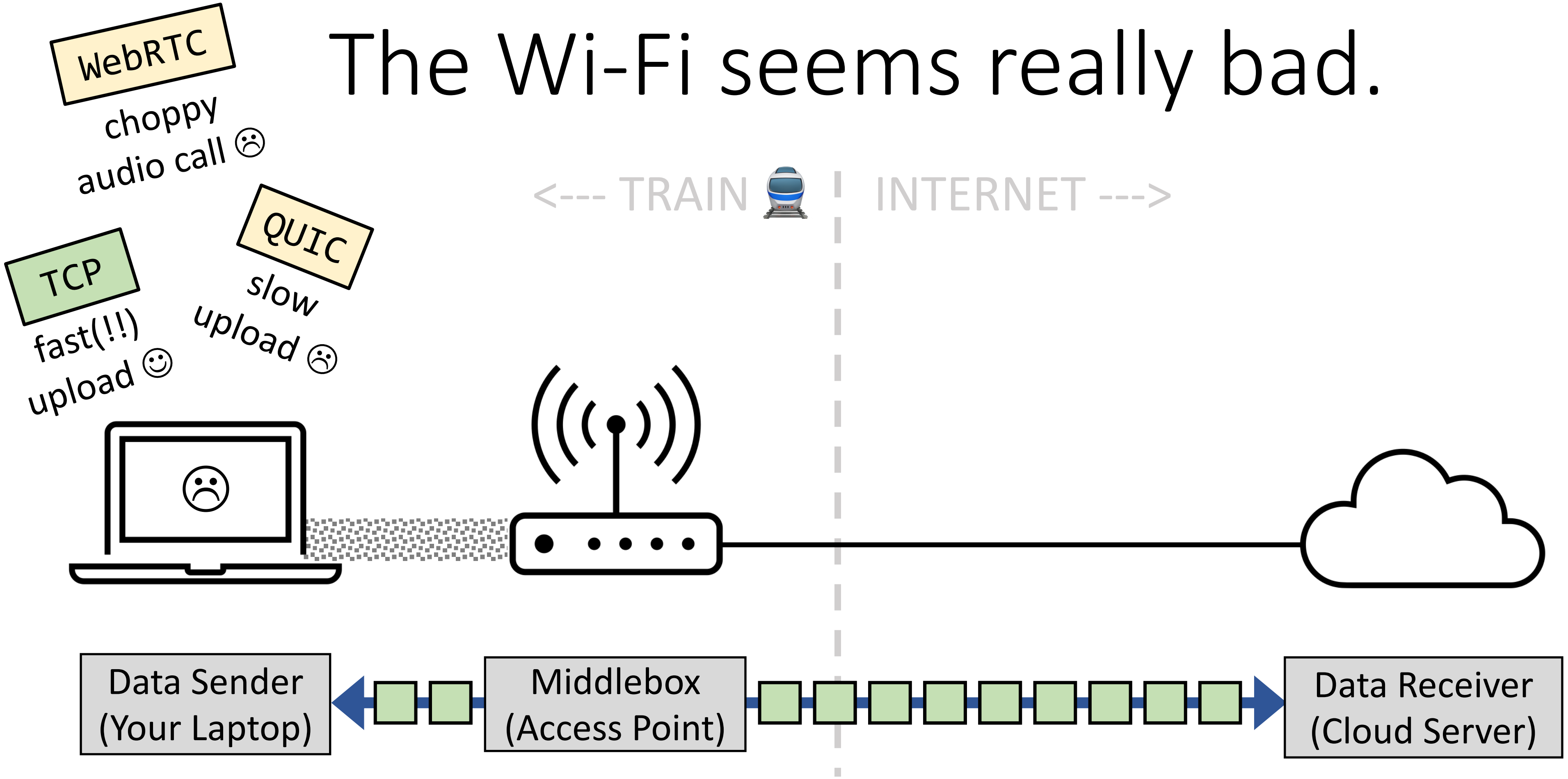
NSDI '24 (April 16-18, 2024)

Stanford University & [+]University of Oslo

Imagine using the Wi-Fi on a train.



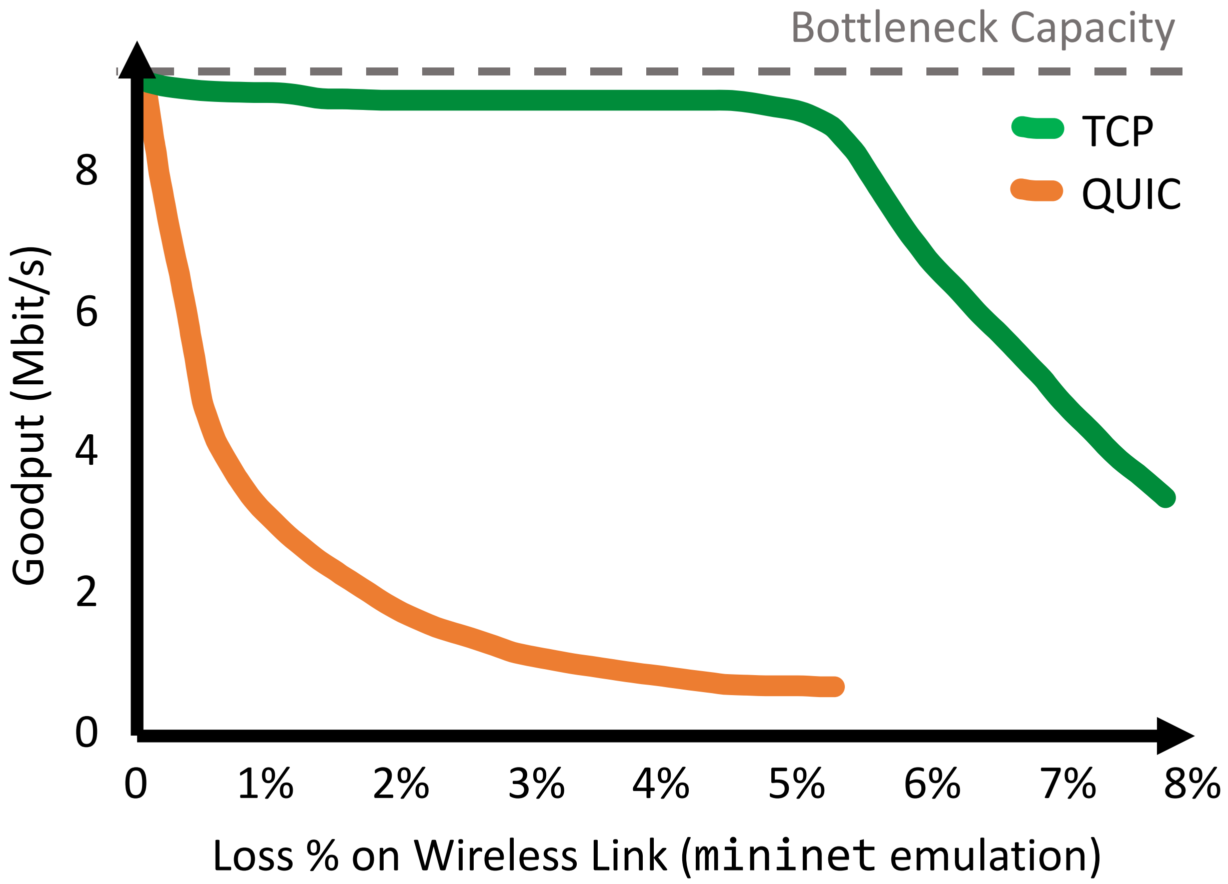
The Wi-Fi seems really bad.

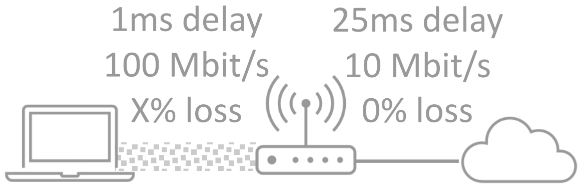


Except for TCP?

TCP is faster than QUIC!?

1970s TCP — Google 2012 QUIC





Experiment Parameters

QUIC (and WebRTC) are *encrypted on the wire.*

QUIC Segment

!@#$%^
(Encrypted Data)

"secure" transport protocol

vs.

TCP Segment

TCP Header (visible to middleboxes)

Source Port			Destination Port
Sequence number			
Acknowledgment number			
DO	RSV	Flags	Window
Checksum			Urgent Pointer
Options			

TCP Payload (encrypted)

TLS-Encrypted Data

TCP is *unencrypted* on the wire.

QUIC Segment

!@#$%^
(Encrypted Data)

"secure" transport protocol

vs.

TCP Segment

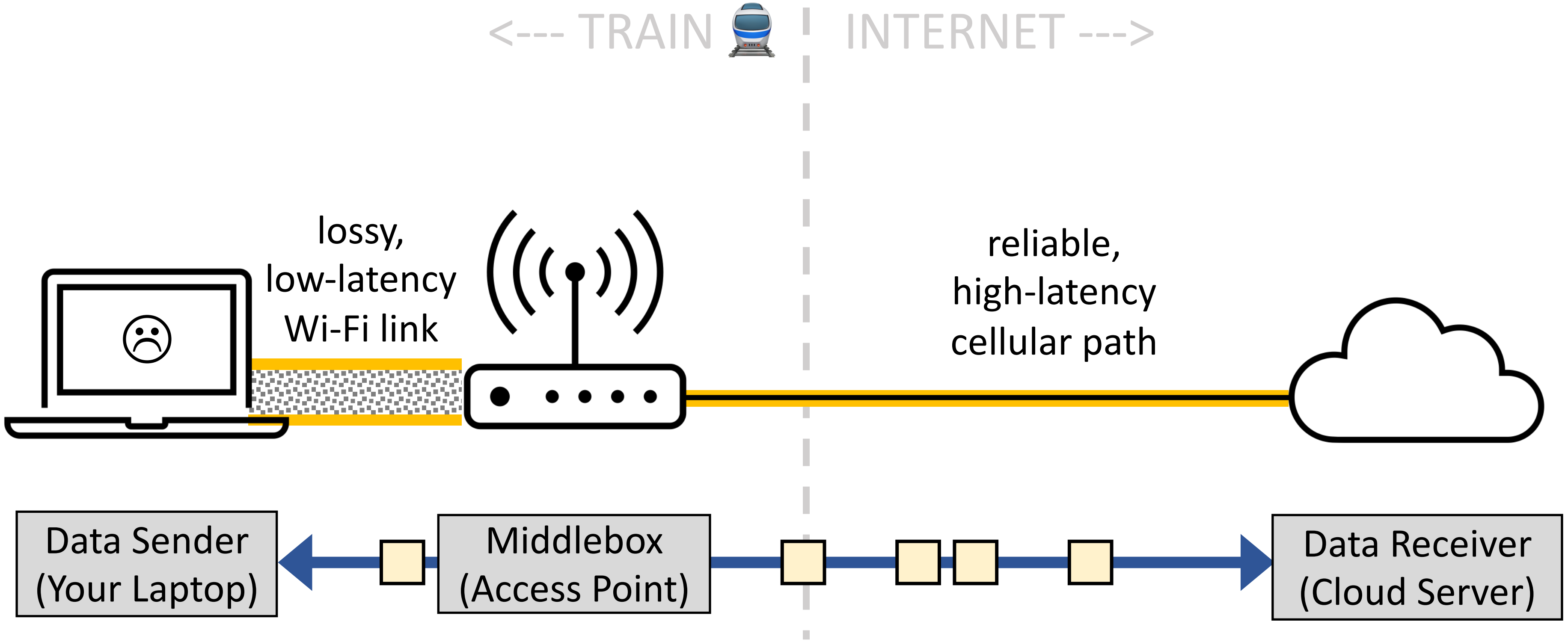
TCP Header (visible to middleboxes)

Source Port			Destination Port
Sequence number			
Acknowledgment number			
DO	RSV	Flags	Window
Checksum			Urgent Pointer
Options			

TCP Payload (encrypted)

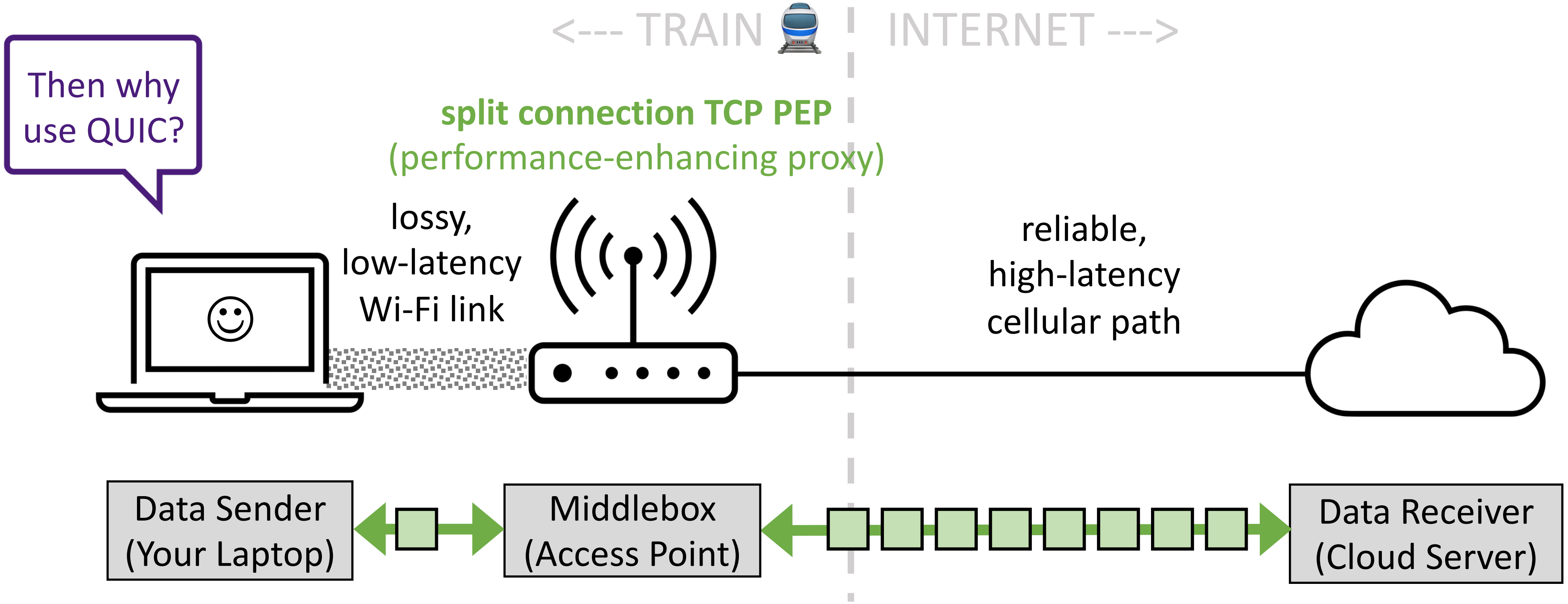
TLS-Encrypted Data

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.



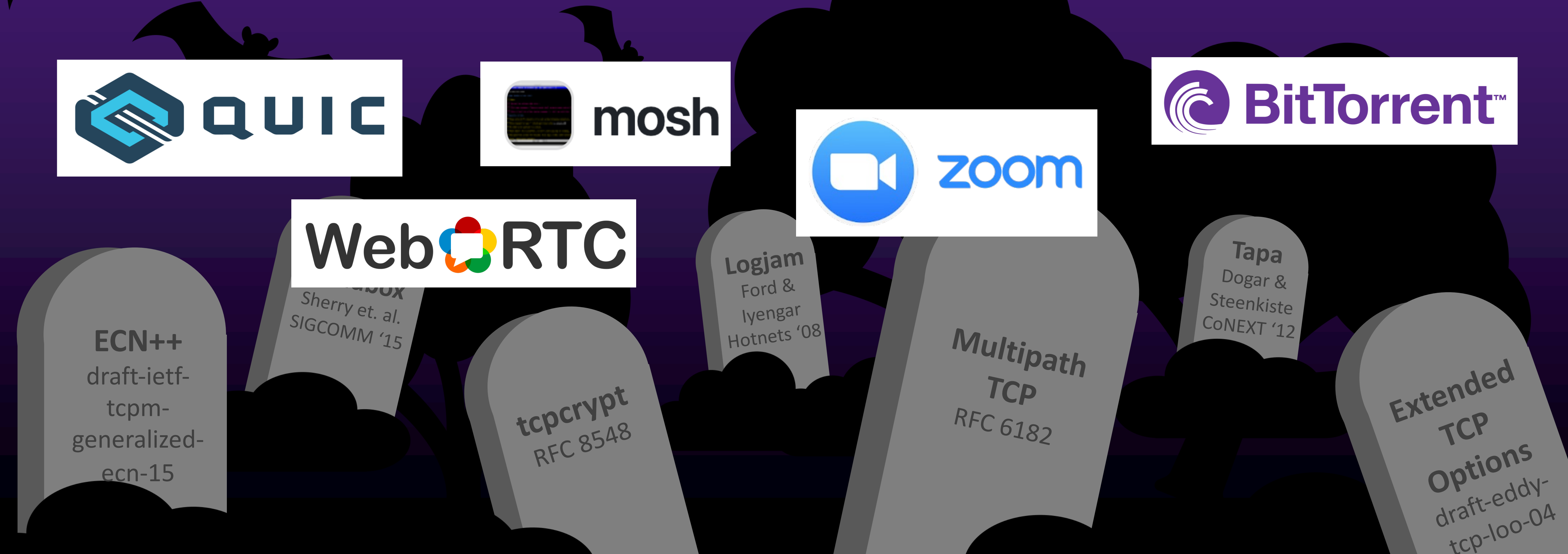
- Faster retransmissions and a better congestion response.
- 20-40% of Internet paths, most cellular paths, contain a TCP PEP [Honda et. al., 2011; Edeline & Donnett, 2019]

Performance-enhancing proxies have a dark side…

- Ossification of existing protocols
- Ossification of future protocols

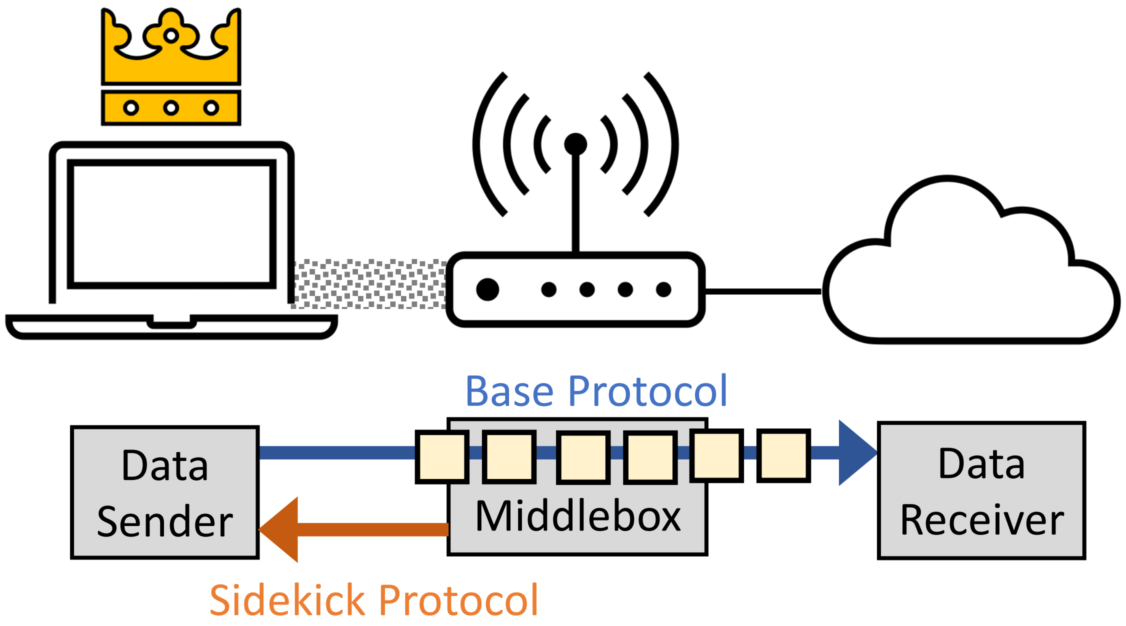
Performance-enhancing proxies have a dark side…

- Ossification of existing protocols
- Ossification of future protocols
- **Today:** encrypt the transport layer (avoiding ossification), but give up on PEPs



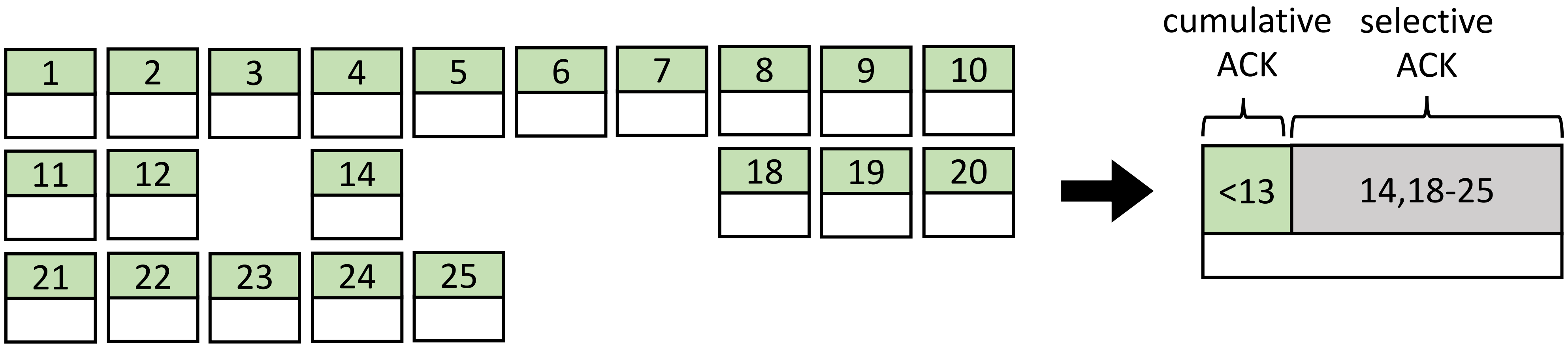
Can there be a universal PEP
for *Arbitrary* transport protocols?

Sidekick protocols: in-network assistance that leaves the base protocol unchanged *on the wire*.



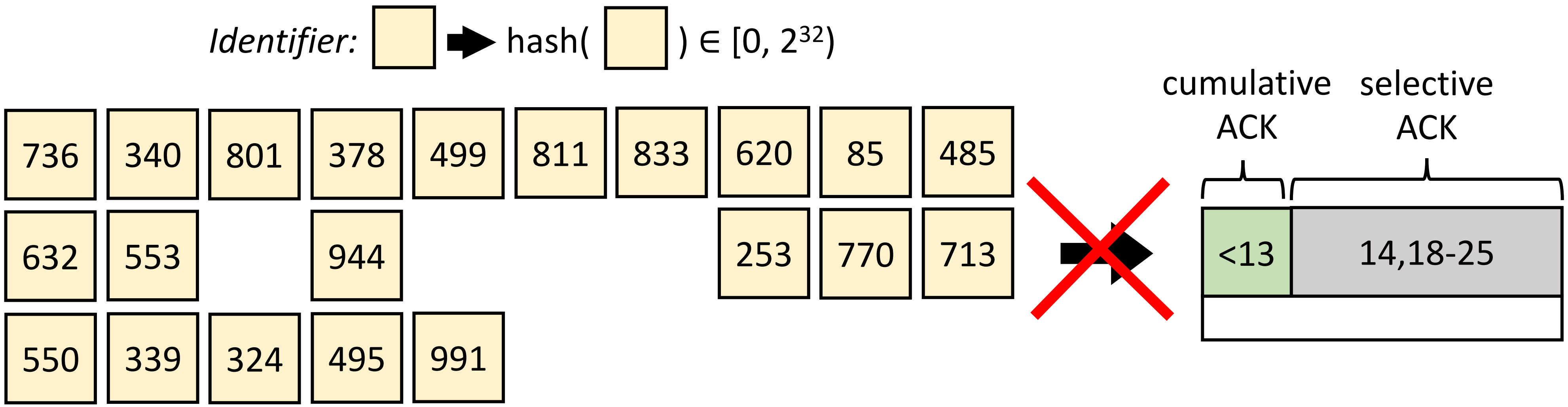
1. Sidekick protocols on an *adjacent* connection

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



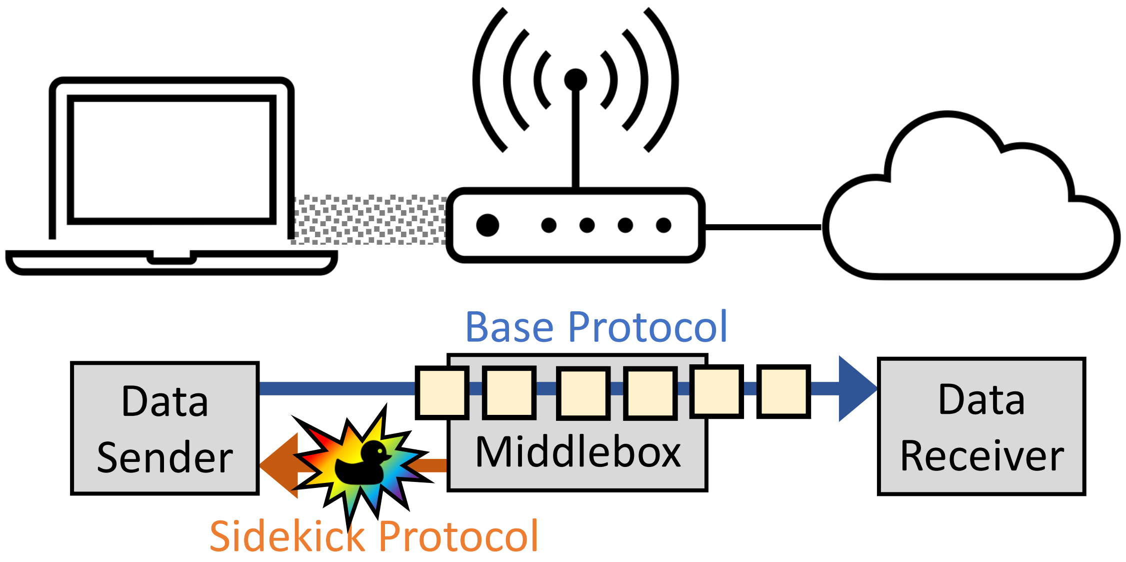
Easy problem for cleartext TCP sequence numbers...

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



But for random-looking packets?

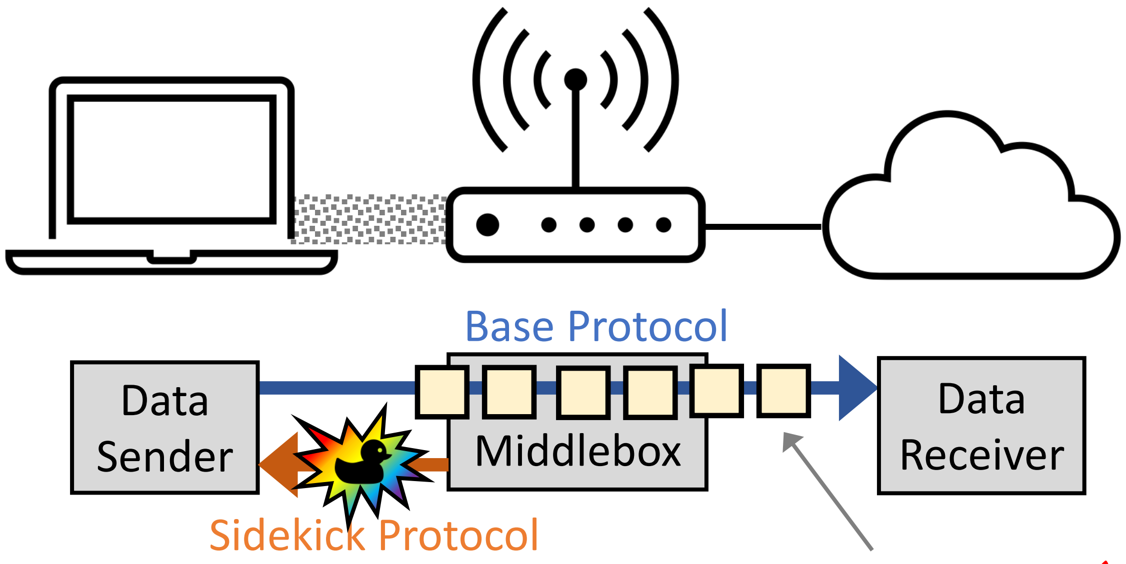
What *useful* information can a middlebox send for random-looking packets?



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

quACK = quick* ACK
*not QUIC ack, since other protocols are ok too!

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



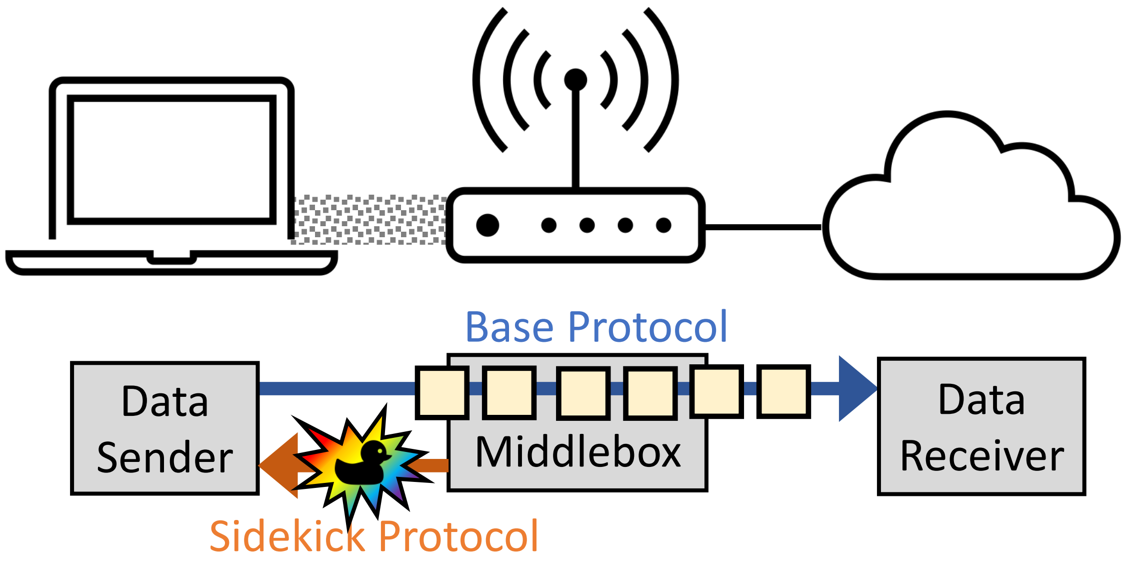
1. Sidekick protocols on an *adjacent* connection
2. QuACKs = concise, efficient ACKs of random packets
3. Path-aware sender behavior: retransmission, congestion control, flow control

Note: *Arbitrary* base protocol.

No reliability guarantee, unlike a TCP ACK.

QuACKs describe *which* packets are received and *where.*

The rest of the talk…



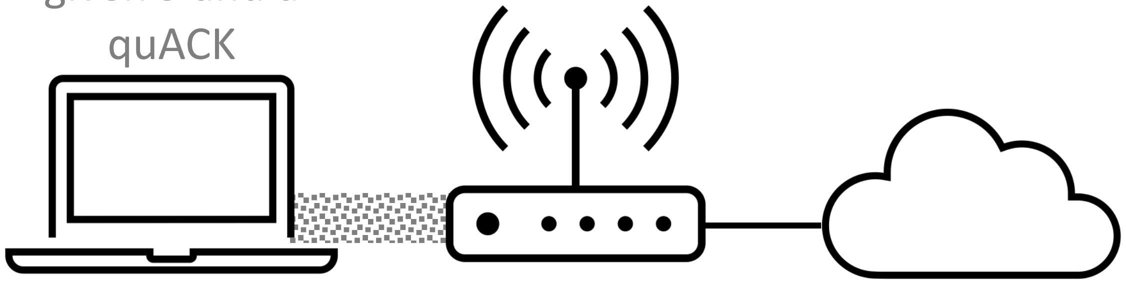
1. Sidekick protocols on an *adjacent* connection
2. QuACKs = concise, efficient ACKs of random packets
3. Path-aware sender behavior: retransmission, congestion control, flow control

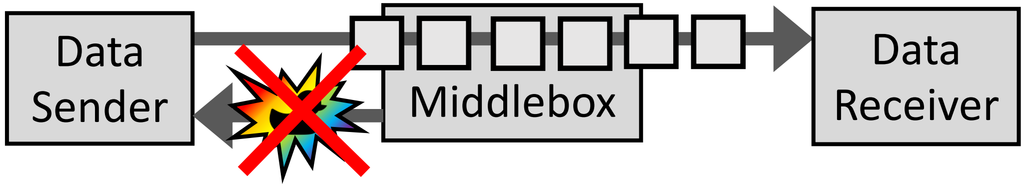
\+ implementation & eval

Desired quACK properties

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

(2a) **Efficient** to process a single packet





quACK = ACK of random packets

(1) **Concise**: low link overhead

(3) **Loss-Resilient**: cumulative representation

How can we construct a quACK with these properties?

	Strawman 1	Strawman 2
Description	Echo every identifier. 736 340 801 378 499 811 833 620 85 485 632 553 944 253 770	Hash a sorted concatenation of every identifier. SHA256(85|...|944)
Encode Time	0	27 ns/pkt
Decode Time	0	830 ms/quACK
QuACK Size	25 pkts × 4 bytes/pkt	36 bytes
Loss-Resilient	No	Yes

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

Power sum solution

[1] David Eppstein and Michael T. Goodrich. 2011. **Straggler Identification** in Round-Trip Data Streams via Newton's Identities and Invertible Bloom Filters. IEEE Trans. Knowl. Data Eng. 23, 2 (2011), 297–306.

	Strawman 1	Strawman 2	Power Sum
Description	Echo every identifier. 736 340 801 378 499 811 833 620 85 485 632 553 944 253 770	Hash a sorted  concatenation of every identifier. SHA256(85|...|944)	Represent missing  identifiers as a system of power sum polynomial equations [1].
Encode Time	0	27 ns/pkt	33 ns/pkt
Decode Time	0	830 ms/quACK	2.82 µs/quACK
QuACK Size	25 pkts × 4 bytes/pkt	36 bytes	48 bytes
Loss-Resilient	No	Yes	Yes

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** ⊆ **S**

threshold t = upper bound on the number of missing packets

n-th power sum = $\sum_{x \in R} x^n$ (finite field)

the last packet received

the number of packets

4*t + 8 bytes

Sender maintains

a threshold number of power sums of the sent packet identifiers **S**

a log of the sent packets

Mathematical Intuition: Decoding QuACKs

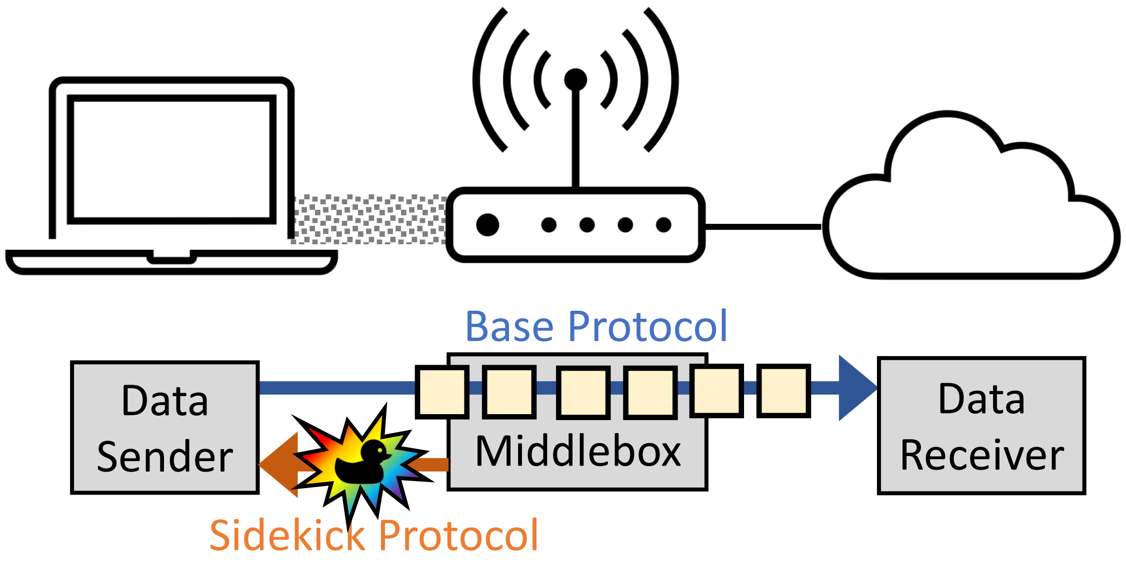
Sender Goal: decode R or S\R given a quACK

one missing packet: $\underbrace{\sum_{x \in S} x}_{\text{sender state}} - \underbrace{\sum_{x \in R} x}_{\text{receiver state}} = \sum_{x \in S \setminus R} x,$ <--- 1 power sum polynomial equation in 1 variable

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

Intuition: Solve a system of *m* polynomial equations in *m* variables, where *m* ≤ a threshold *t*. The solutions are the missing packets.

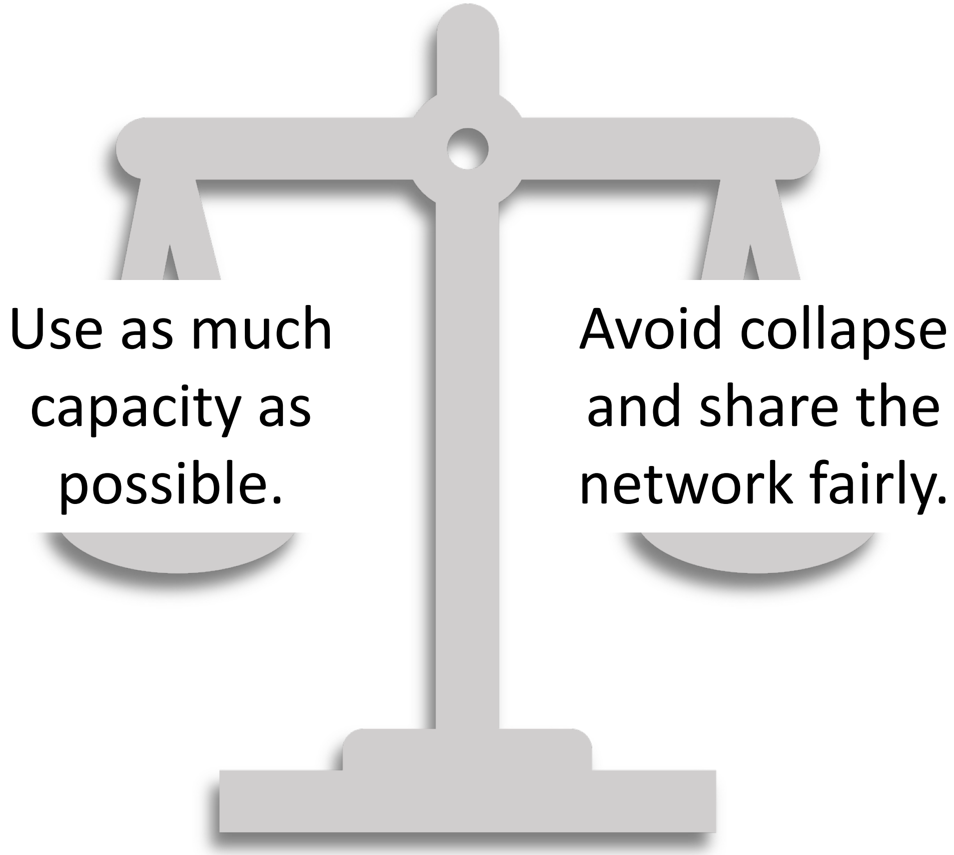
Talk Outline



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

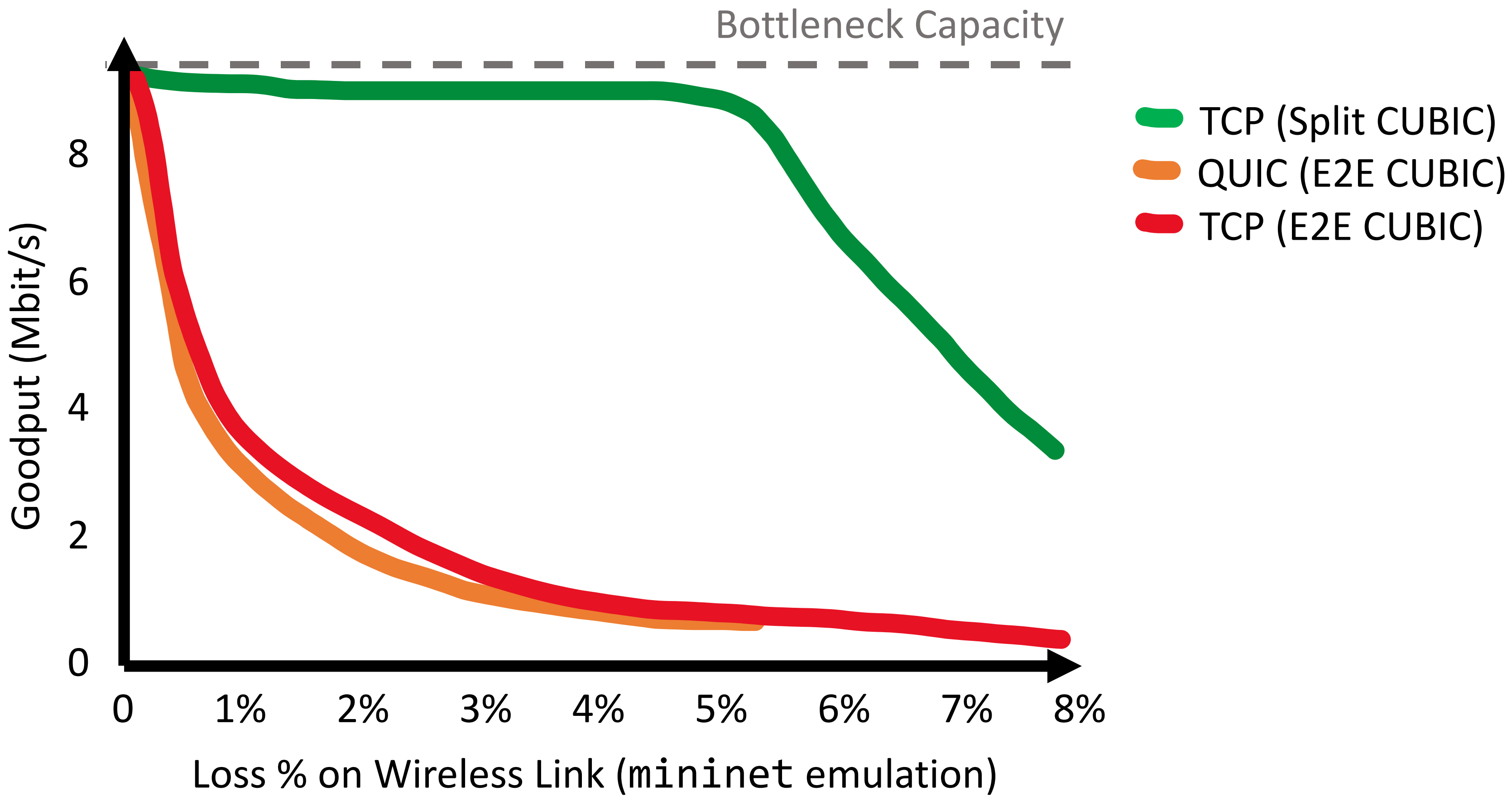


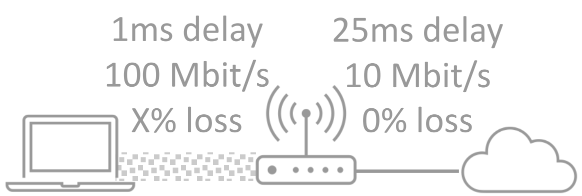
CUBIC modulates a *congestion window* (cwnd), using loss from ACKs as a signal.

From before: QUIC and TCP both used CUBIC.

one end-to-end CUBIC cwnd

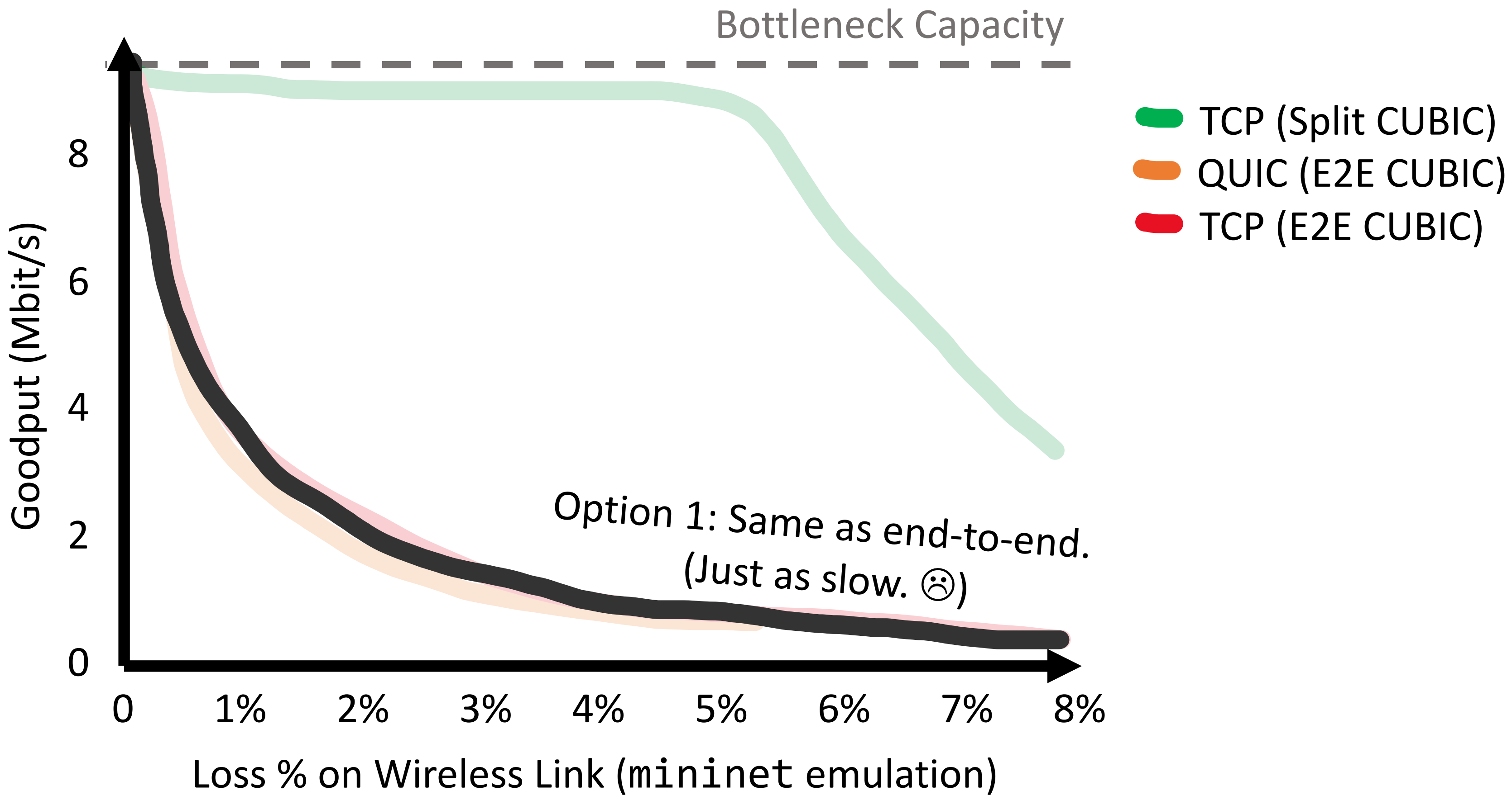
two connections, each with its own CUBIC cwnd

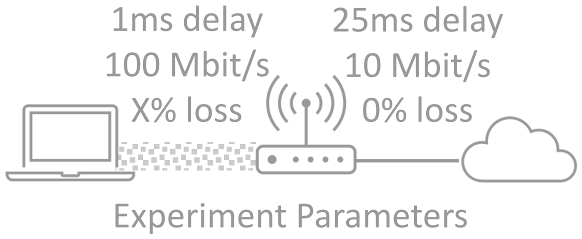




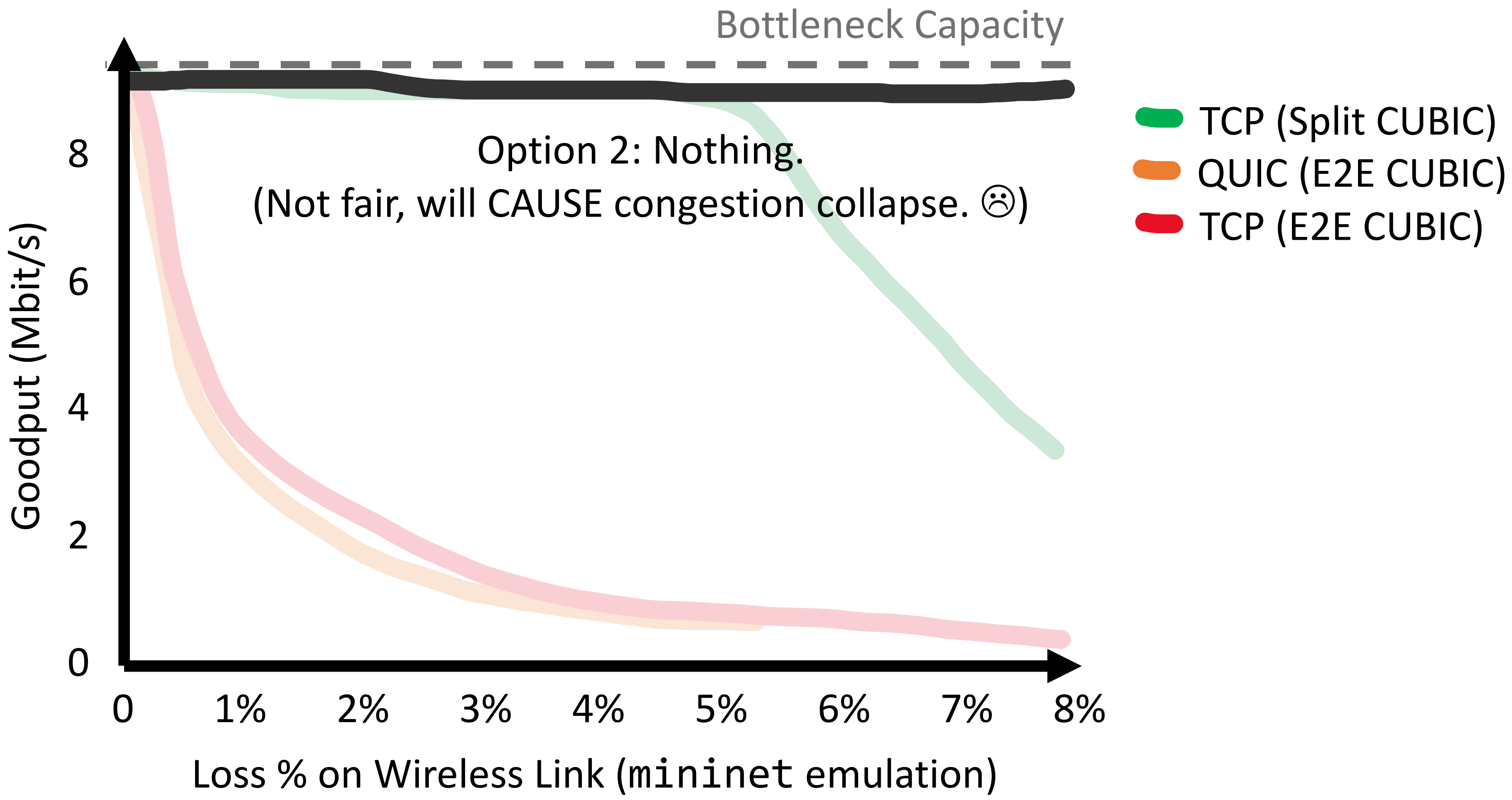
Experiment Parameters

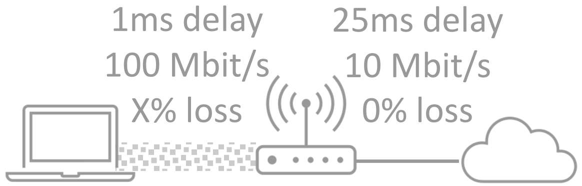
What should be the congestion response to loss from *quACKs* to obtain a performance benefit?





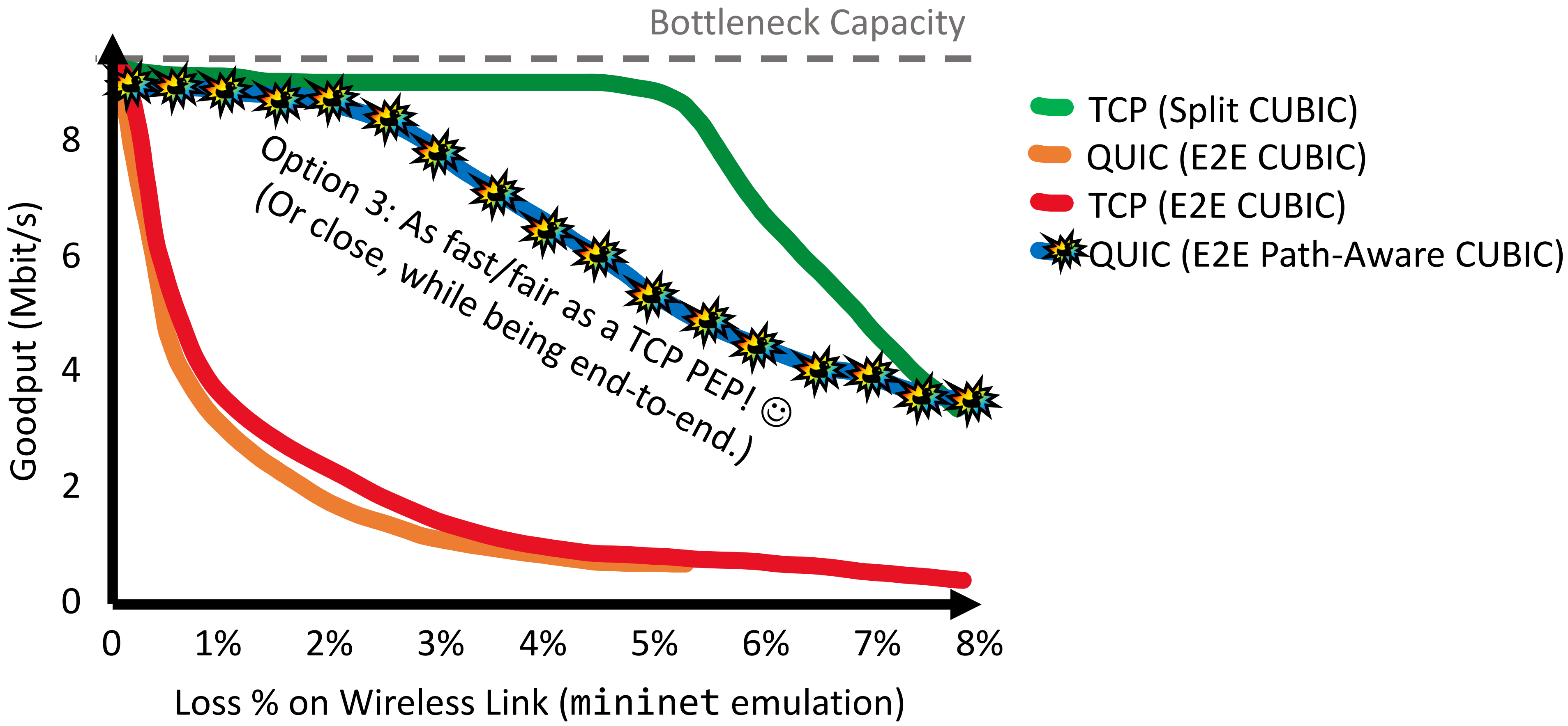
What should be the congestion response to loss from *quACKs* to obtain a performance benefit?

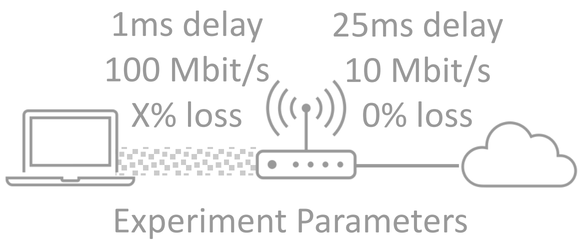




Experiment Parameters

What should be the congestion response to loss from *quACKs* to obtain a performance benefit?





Mathematical Intuition: Path-aware CUBIC

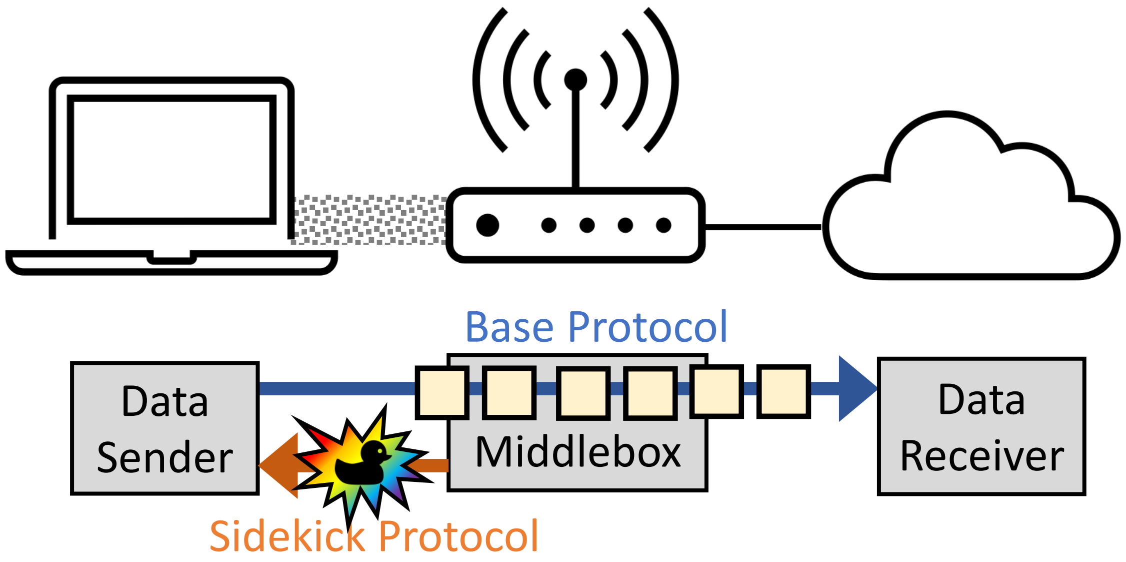
Idea: Update the portion of the end-to-end cwnd that corresponds to the path segment of the last congestion event.

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
β = 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

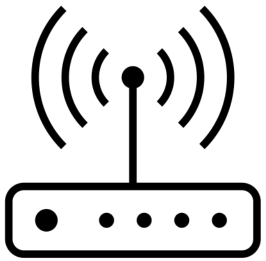
Module	Language	LOC
QuACK library	Rust	1772
Middlebox sidekick binary	Rust	833
`quiche` client integration	Rust	1821
`libcurl` client integration	C	1459
Media server/client + integration	Rust	478

https://github.com/ygina/sidekick

NSDI '24 Community Award!

Middlebox binary

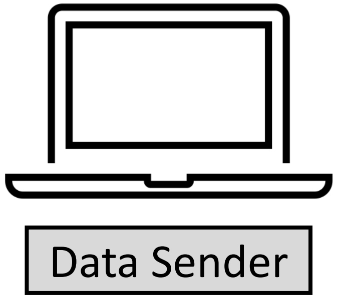
- **74 cycles/pkt** (0.9%) to encode an identifier into a *t=10* quACK
- Largest overhead was reading the packet contents from the network interface (97.5% of cycles/pkt)
- Max achieved throughput for a single core was **464k pkts/s** on a 2.30 GHz CPU



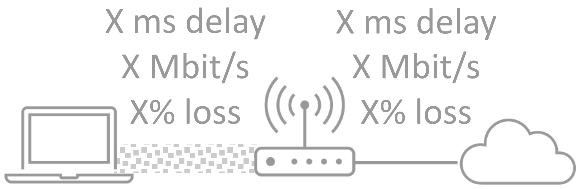
Middlebox

Client integrations

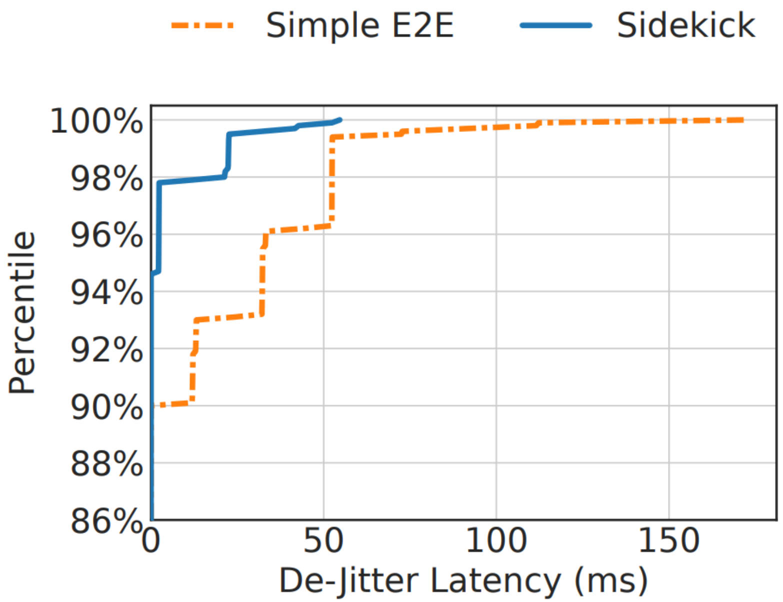
- Simple media client/server
  - ~150 additional LOC (Rust) to utilize sidekick protocols for **retransmission**
- QUIC+HTTP/3 production client/server
  - ~1500 additional LOC (C) to establish sidekick connection in `libcurl` client
  - ~1800 additional LOC (Rust) to implement **retransmission**, **congestion control**, and **flow control** logic in Cloudflare `quiche`
  - **Overhead:** 3% more packets, quACKs and ACKs have similar processing time



Applications (Emulation)

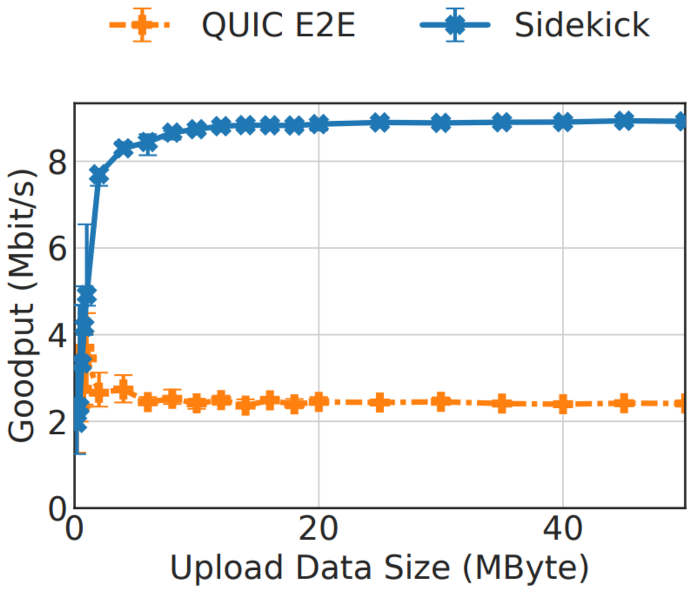


Various Parameters (see paper)



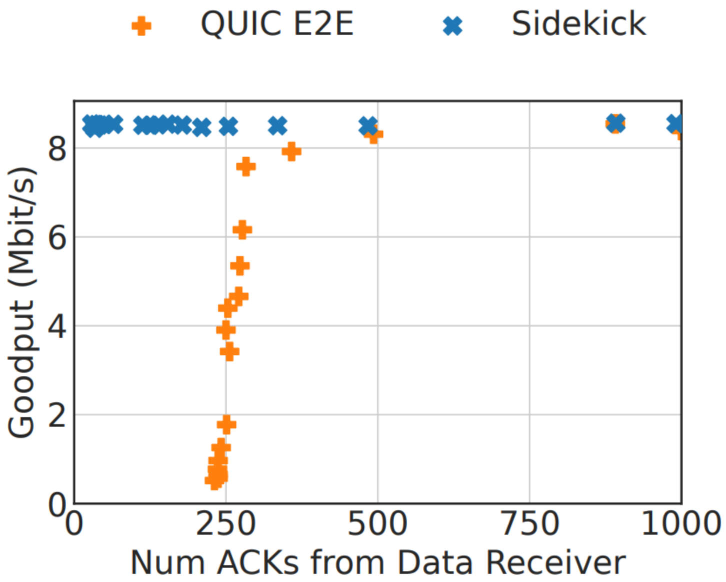
Scenario #1: Low-Latency Media (simple media protocol)

Goal: Reduce de-jitter buffer **tail latency**.



Scenario #2: Connection-Splitting PEP Emulation (HTTP/3+QUIC)

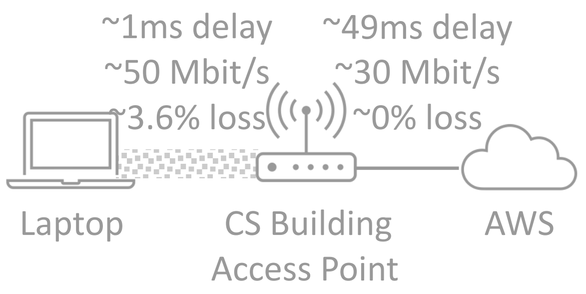
Goal: Achieve high **throughput** while being as fair as TCP PEPs.

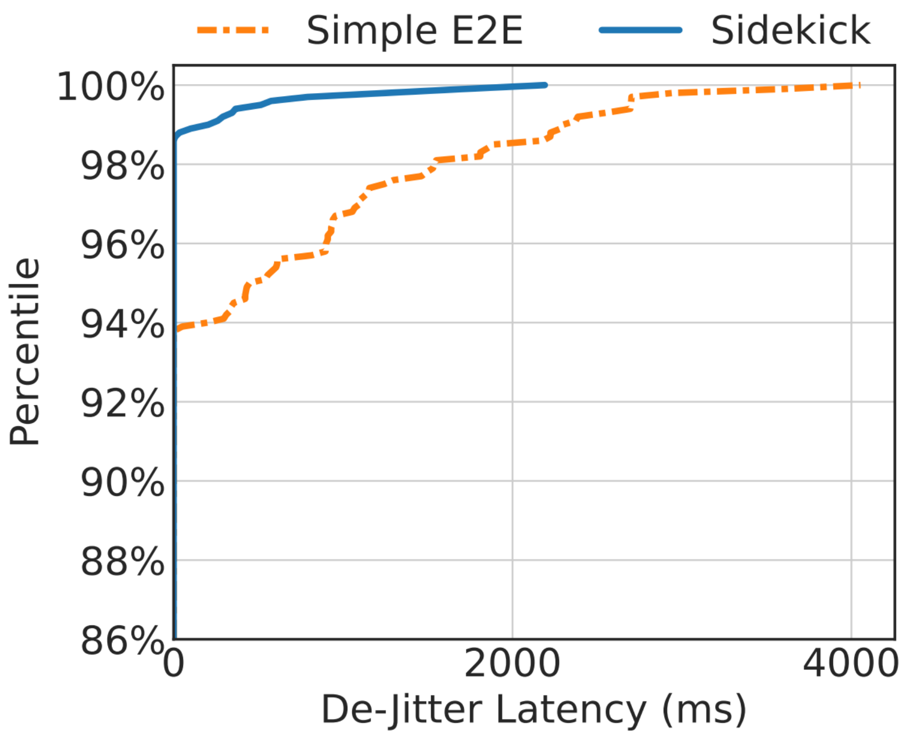


Scenario #3: ACK Reduction (HTTP/3+QUIC)

Goal: Reduce the **ACK frequency** of a receiver to save energy.

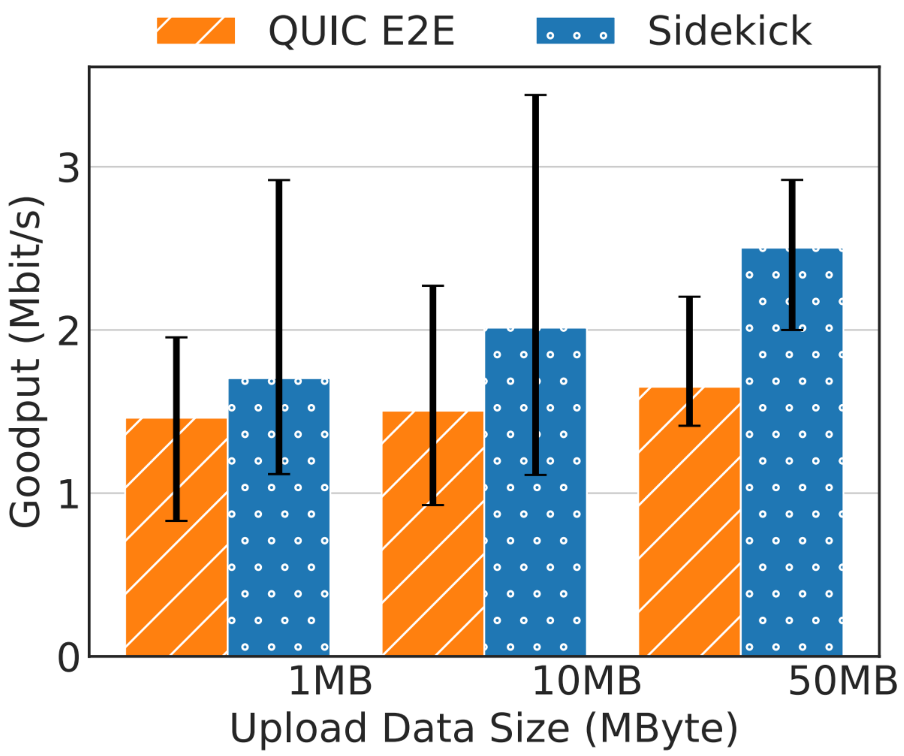
Applications (Real World)





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/

Gina Yuan

Matthew Sotoudeh

David K. Zhang

Michael Welzl

David Mazières

Keith Winstein